PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 :
H04L 12/56

A1

(11) International Publication Number: WO 00/10297

(43) International Publication Date: 24 February 2000 (24.02.00)

(21) International Application Number: PCT/US99/18665

(22) International Filing Date: 16 August 1999 (16.08.99)

(30) Priority Data:

60/096,771	17 August 1998 (17.08.98)	US
60/099,489	8 September 1998 (08.09.98)	US
60/133,241	7 May 1999 (07.05.99)	US
09/373,800	13 August 1999 (13.08.99)	US

(71) Applicant *(for all designated States except US)*: VITESSE SEMICONDUCTOR CORPORATION [US/US]; 741 Calle-plano, Camarillo, CA 93012 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: DEB, Alak, K. [US/US]; 3230 Vintage Crest Drive, San Jose, CA 95148 (US). SAMBAMURTHY, Namakkal, S. [IN/US]; 3408 Casalino Court, San Jose, CA 95148 (US). TRIPATHI, Devendra, K. [IN/US]; 1825 Canal Way, San Jose, CA 95132 (US).

(74) Agent: OLYNICK, Mary, R.; Beyer & Weaver, LLP, P.O. Box 61059, Palo Alto, CA 94306 (US).

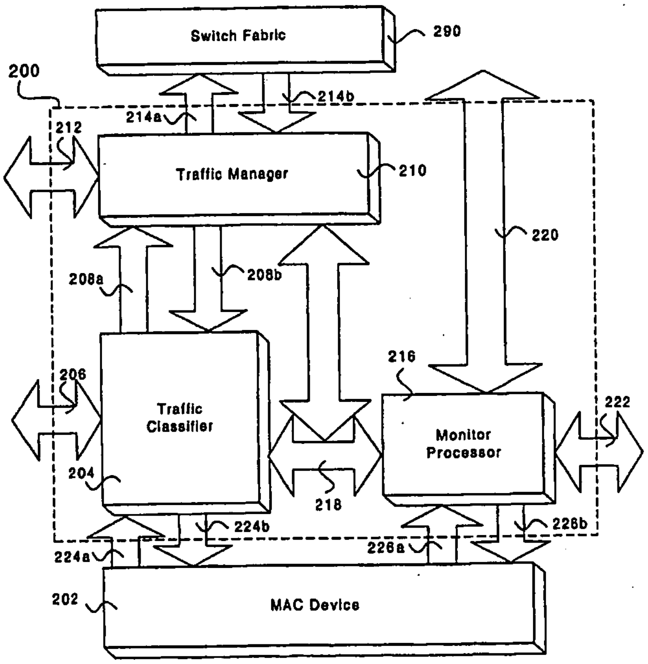
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: PACKET PROCESSING ARCHITECTURE AND METHODS

(57) Abstract

A packet processing system (153, 200, 280) for receiving and processing a data packet is disclosed. The processing includes a plurality of tasks and a packet processing system associated with a switching device for transmitting the packet after it is processed by the packet processing system to another network device. The packet processing system includes a first processor (156, 158, 160, 162, 165, 156, 174, 176) arranged to perform a first subset of tasks on the data packet and associate a first header with the packet. The first header is based on information obtained while performing the first subset of tasks. The packet processing system also includes a second processor (156, 158, 160, 162, 165, 156, 174, 176). The second processor is arranged to perform a second subset of tasks which are associated with the packet. The second subset of tasks are based on the first header which is supplied by the first processor. The first subset of tasks are different from the second subset of tasks.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

PACKET PROCESSING ARCHITECTURE AND METHODS

BACKGROUND OF THE INVENTION

The present invention relates generally to integrated circuit devices used for processing data through communication networks, and more particularly, to methods and apparatuses for managing and processing the flow of data in high speed networks.

A computer network typically includes a set of communication channels interconnecting a set of computing devices or nodes that can communicate with each other. These nodes may be computers, terminals, workstations, or communication units of various kinds distributed over different locations. They communicate over communications channels that can be leased from common carriers (e.g. telephone companies) or may provided by the owners of the network. These channels may use a variety of transmission media, including optical fibers, coaxial cable, twisted copper pairs, satellite links or digital microwave radio. The nodes maybe distributed over a wide area (distances of hundreds or thousands of miles) or over a local area (distances of a hundred feet to several miles), in which case the networks are called wide area (WAN) or local area (LAN) networks, respectively. Combinations of LANs and WANs are also possible by coupling widely separated LANs, for example in branch offices, via a WAN.

Recently, computer networks have greatly increased in number and geographical area. A modern network may consist of thousands of computing devices made by various manufacturers connected by a variety of transmission media spanning international and intercontinental boundaries. As the complexity of networks has increased, there have also been continuing efforts to increase network performance in terms of the speeds that data traffic may be transmitted over a given network. To accommodate the needs of increased network sizes, complexities and speeds, a wide variety of networking products have been developed. One class of products that have been developed are generally referred to as switching devices, which are used to route data packets within and between networks and network nodes. These switching devices take a wide variety of forms and include routers, switching hubs, bridges, and repeaters. and remote monitoring probes.

Referring initially to Figure 1A, a conventional multi-ported switch system will be briefly described. As seen therein, a switch fabric 102 is typically coupled to various links or network branches by ports 100. Each port typically includes an interface module 103, memory 104, a media access controller (MAC) 110 and a physical layer controller 112. The physical layer controller 112 is responsible for encoding and decoding data into signals that are transmitted across a particular link medium (e.g. wire, fiber, air, etc.). The media access controller 112 is responsible for scheduling, transmitting and receiving data over its associated link. Thus, MAC 110 is primarily responsible for controlling the flow of data over the link, ensuring that transmission errors are detected, and ensuring that transmissions are appropriately synchronized. The interface module 103 is an intermediary between its associated MAC 110 and the switch fabric 102 and provides a variety of functions. One function of the interface module is to queue packets received from the MAC 110 and schedule their transmission into the switch fabric 102. Other functions may involve various levels of packet processing, filtering and/or editing. Additionally, the interface module receives packets from the switch fabric and passes them to the associated MAC to schedule for transmission over the associated link. The switch fabric 102, itself, can be implemented as a crossbar interconnection device, as a shared bus, or as a shared memory.

As data transmission speeds increase and the variety of protocols required to represent and interpret the data increase, conventional methods of packet processing typically need to be reevaluated in order to facilitate performance. As will be appreciated by those skilled in the art, although the design of the interface modules used in various switching devices vary extensively, most rely on the use of a central processing unit (CPU) to perform most of the required functions. Although conventional interface modules have worked well in the past, as higher speed systems are implemented, they generally impose high processing burdens on the interface module's host's central processing unit (CPU). By way of example, when Ethernet network speeds are accelerated to gigabit levels, the host CPU will generally be required to spend more time processing packet data and less time performing other CPU processing tasks. As a result, the host CPU will tend to experience many more processing interrupts which may hamper packet transmission and receiving operations.

As an example, when packet data is received from the MAC layer 110, the CPU is conventionally required to scan through each and every bit of data in the order received to locate the byte location of headers and data that may be of interest when routing, prioritizing and/or otherwise processing the packets. Once the CPU has searched the packet to obtain the desired information, typically one or more lookups of routing, forwarding, or filtering information from a database of a large number of entries are performed to obtain where the packet needs to be sent and how it needs to be handled. The information may then be used to accomplish the desired routing, prioritization and/or processing. Unfortunately, in high speed networks, the demands on a host CPU tends to increase to levels where scanning through each byte of each received packets is no longer possible without introducing decoding, transmission or packet routing delays.

In addition to the CPU processing performed during the receiving of packet data, the host CPU is also charged with the responsibility of analyzing each byte of an outgoing packet. By way of example, when the host is a switch or a router, the switch or router CPU is generally responsible for managing routing tables, and analyzing flow congestion. By way of another example, a router CPU is typically responsible for flow classification, policy based routing determinations, complex protocol and signature based filtering, and packet editing. In addition, the switch or router CPU is also responsible for building and managing routing tables in an effort to constantly update the status of each and every node in the network. Other host CPU tasks may include performing management tasks, replying to queries from management hosts, building remote monitoring (RMON) data bases, etc. Accordingly, when a network host is asked to transmit data at increased speeds, the host CPU will unfortunately be more prone to CPU interrupt related delays.

Some solutions have included hardware that is wired to perform specific processing at very high speeds. For example, the internet has a large number of packet formats and protocols that need to be supported, and this number continues to increase as various standards bodies, such as the IETF and the IEEE, continuously generate different kinds of headers and encapsulation schemes to meet the needs of an ever growing number of network applications and services. Such dynamics of the industry make it impossible for such "hard wired" solutions to continue to be useful.

In view of the foregoing, there is a need for methods and apparatuses for data processing that are well suited to increase transmit and receive packet processing rates while reducing a host CPU's processing burden. Additionally, more flexible mechanisms for efficiently modifying data processing schemes are desired.

SUMMARY OF THE INVENTION

Broadly speaking, the present invention fills these needs by providing apparatus and methods for a high speed packet processing system that includes processors for performing various data processing tasks. The packet processing system may be configured to perform any number and kind of tasks on data packets. Preferably, the tasks are performed serially on the packets as they are streamed (*e.g.*, pipelined) through the packet processing system.

In one embodiment, a packet processing system for receiving and processing a data packet is disclosed. The processing includes a plurality of tasks, and the packet processing system is coupled with a switching device for transmitting the packet after it is processed by the packet processing system to another network device. The packet processing system includes a first processor arranged to perform a first subset of tasks on the data packet and associate a first header with the packet. The first header is based on information obtained while performing the first subset of tasks. The packet processing system further includes a second processor arranged to perform a second subset of tasks for the packet based on the first header supplied by the first processor. The first subset of tasks differ from the second subset of tasks.

In several embodiments, the packet processing system includes a traffic classifier that is arranged to receive an incoming packet and associate information with the received packet. This associated information is indicative of or useful in at least one of routing and prioritization of the received packet. In some preferred embodiments, the traffic classifier includes a first stream processor that is arranged to perform a first subset of tasks on the received data packet and associate a first header with the packet. The first header is based on information obtained while performing the first subset of tasks. The traffic classifier further includes a queue structure arranged to receive the packet, and a search engine suitable for looking up data which is associated with information in the packet based on the first header. Based on the search results, the search engine may also create a second header that is associated with the packet. The traffic classifier may also include a post editor for performing a second subset of tasks on the processed data packet. In various embodiments, the packet processing system further includes a traffic manager that communicates with the switch fabric. Structurally, the traffic manager includes a large number of queues and is arranged to manage these queues. More specifically, the traffic manager is arranged to receive packets from the traffic classifier and insert them in the appropriate queue based at least in part upon the information associated with the packet by the traffic classifier. In a preferred embodiment, the traffic classifier and traffic manager are programmable.

In an alternative embodiment, a packet processing system for receiving and processing a data packet from a media access control device or framer is disclosed. The packet processing system is coupled with a switching device for transmitting the packet to another network device after it is processed by the packet processing system. The packet processing system includes a programmable traffic classifier arranged to perform a plurality of traffic classification tasks on the packet (such as classifying the packet, filtering the packet, encrypting or decrypting the packet, editing the packet, and scheduling the packet). The first processor is also arranged to attach a data manipulation header to the packet. The packet processing system also includes a traffic manager arranged to receive the packet and perform a plurality of traffic management tasks on the packet (such as providing a plurality of queues for routing the packet, quality of service tasks, network scheduling, congestion handling, and queue reordering functions) The management tasks re based on the data manipulation header. In a preferred embodiment, the classification, management, and monitoring tasks are individually programmable.

In another embodiment, a packet processing system includes a traffic classifier arranged to receive an incoming packet and associate classifier information with the received packet. The classifier information is indicative of or useful in at least one of desired routing and prioritization of the received packet. The traffic classifier includes a first stream processor arranged to perform a first set of tasks on the received data packet and to associate a first header with the packet, the first header being based on information obtained while performing the first subset of tasks, a queue structure arranged to receive the packet, a search engine for looking up data associated with information in the packet based on the first header and create a second header that is associated with the packet in the queue structure. The packet processing system also includes a traffic manager having a multiplicity of queues associated therewith for queuing packets to be transmitted to another device. The traffic manager is arranged to receive packets from the traffic classifier and insert them in an appropriate queue based at least in part upon the classifier information associated with the packet by the traffic classifier.

In yet another packet processing system embodiment, the system includes a traffic classifier arranged to receive an incoming packet and associate information indicative of or useful in at least one of routing and prioritizing the packets. The traffic classifier includes a stream processor arranged to extract information from the packet and a search engine arranged to look up data based on information extracted by the stream processor and to associate such looked up data with the packet. The system also includes a traffic manager having a multiplicity of queues associated therewith for queuing packets to be transmitted to another device. The traffic manager is arranged to receive packets from the traffic classifier and insert them in an appropriate queue based at least in part upon the information associated with the packet by the traffic classifier.

In another aspect, the invention pertains to a router that includes a switching device and a plurality nodes that are coupled to the switching device. Each node includes a packet processing system as described above in any of the embodiments, a media access controller, and a physical layer controller.

In another embodiment, a method for processing packet data received from a media access layer is disclosed. The processing is performed in-line while streaming packets to an upper layer. An instruction set for custom programming the processing of packet data received from the media access layer is loaded. The packet data is received within a first processor. Information is extracted from the packet data based at least in part on the instruction set, and a first data structure is created based at least in part on the extracted information. The first data structure is associated with the packet data before the packet is streamed to a second processor. Information is extracted from one or more databases based on the first data structure. A second data structure is created based at least in part on the first data structure and information extracted from the one or more databases. The second data structure is associated with the packet data before the packet is streamed to a third processor. The packet data is received within the third processor having a plurality of queues for routing data to various devices, and The data packet is linked to a particular queue of the third processor based at least in part on the second data packet.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures which illustrate by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figure 1A is a diagrammatic representation of a conventional multi-ported switch system that includes a switch fabric coupled to various links or network branches by ports.

Figures 1B and 1C generally illustrate two possible configurations of a packet processing system that is suitable for replacing the interface module of Figure 1A in accordance with one embodiment of the present invention.

Figure 2A is an architectural diagram of a packet processing system for in accordance with one embodiment of the present invention.

Figure 2B is an architectural diagram of a packet processing system in accordance with an alternative embodiment of the present invention.

Figure 3 is an architectural diagram of a router that employs a plurality of the packet processing systems of Figure 2A or 2B.

Figure 4 is a diagrammatic representation of a traffic classifier suitable for use in the packet processing systems of Figures 2A and 2B in accordance with one embodiment of the present invention.

Figure 5A is a high level block diagram illustrating the interactions between a host CPU and a bit stream processor suitable for use in the traffic classifier of Fig. 4 in accordance with one embodiment of the present invention.

Figure 5B is an architectural diagram of an implementation of the bit stream processors of Fig. 4 in accordance with one embodiment of the present invention.

Figure 6 is an overview flowchart diagram of the operations performed within the bit stream processors of Figs. 5A and 5B in accordance with one embodiment of the present invention.

Figure 7 is a diagrammatic representation of the search engine of Fig. 4 in accordance with one embodiment of the present invention.

Figure 8 is a diagrammatic representation of a linked list database in accordance with one embodiment of the present invention.

Figure 9 is a diagrammatic representation of a variable match size database in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Reference will now be made in detail to specific embodiments of the invention. While the invention will be described in conjunction with specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

As described above with reference to Figure 1A, a conventional multi-ported switch system typically includes a switch fabric 102 that is coupled to various links or network branches by ports 100. Each port typically includes an interface module 103, memory 104, a media access controller (MAC) 110 and a physical layer controller 112. The physical layer controller 112 is responsible for encoding and decoding data into signals that are transmitted across a particular link medium (e.g. wire, fiber, air, etc.). The media access controller 112 is responsible for scheduling, transmitting and receiving data over its associated link. Thus, MAC 110 is primarily responsible for controlling the flow of data over the link, ensuring that transmission errors are detected, and ensuring that transmissions are appropriately synchronized. The interface module 103 is an intermediary between its associated MAC 110 and the switch fabric 102 and provides a variety of functions. One function of the interface module is to queue packets received from the MAC 110 and scheduling their transmission into the switch fabric. Other functions may involve various levels of packet processing, filtering and/or editing. Additionally, the interface module receives packets from the switch fabric and passes them to the associated MAC to schedule for transmission over the associated link.

Figures 1B and 1C generally illustrate two possible configurations of a packet processing system 153 that is suitable for replacing the interface module 103 of Figure 1 in accordance with one embodiment of the present invention. The processing system 153 may be configured to serially perform any number and kind of tasks on a packet. Tasks may be selected from a library of tasks associated with the processing system 153, and/or custom tasks may be created and added to the tasks selected from the library. Also, the order in which tasks are performed may also be selected and preconfigured (*e.g.* as shown in Figures 1B and 1C).

As shown in Figure 1B, the processing system 153 is configured to process received packets 154 through a series of dedicated task modules. In the illustrated embodiment, packets 154 are received from a network, processed through the task modules, and transmitted to a fabric. The task modules may be configured to perform any number and type of tasks. As shown, the task modules include a parse module 156, a search module 158, a first edit module 160, a decryption module 162, a second edit module 165, and a schedule module 165.

The parse module 156 generally parses through the packet contents to locate relevant information that may be used by later modules, such as the search module 158. The search module 158 may then perform a search within a data base for information related to the packet. This search may be based, in part, on the parsed information. The first edit module 160 may then edit the packet based on the information found by the search module 158 and/or the by the parse module 156. The decrypt module 162 may then perform a decryption algorithm that may or may not be based on the information from the search module 158 and the parse module 156. The second edit module 164 may then perform further edits on the packet based on the results of the tasks performed from any preceding module. The schedule module 165 determines when and how to route each packet, and these routing determinations may be based on information provided by preceding task modules.

Preferably, the tasks of each module are performed in an assembly line manner on packets as they serially progress from task module to task module. That is, a first packet may be processed by the parse module 156 and then passed to the search module 158. While the first packet is being processed by the search module 158, a second packet may be received and processed by the parse module 156. As more packets are received, it is possible that all task modules are simultaneously performing tasks on various packets. This pipelining of tasks provides an efficient mechanism for processing packets.

Likewise, as shown in Figure 1C, the processing system 153 may be configured to serially perform tasks on a packet prior to transmission. These tasks may include a multi-cast search within a database (172), editing of the packet (174), and encrypting of the packet (176) prior to transmission through line 178. Of course, the configured tasks of Figures 1B and 1C are merely illustrative, and, of course, tasks may be added to or deleted from the illustrated task modules.

Referring next to Figure 2A, a packet processing system 200 that is suitable for use as an interface module in accordance with one embodiment of the present invention will be described. The packet processing system 200 includes a programmable traffic classifier 204, a programmable traffic manager 214, and a programmable monitor processor 216. The traffic classifier 204 receives incoming packets from an appropriate source. In general terms, the traffic classifier 204 is arranged to peek into a received packet and perform a set of programmed tasks that are intended to provide information about the received packet that is thought to be useful in handling the packet downstream. The traffic classifier information is then associated with the packet and made available to the traffic manager 214 in a form suitable for use by the traffic manager 214 (or other device). In some situations the information is read directly from the packet, while in others cases the classifier performs look up operations, edits and/or manipulates packet information.

The specific tasks performed by the traffic classifier 204 may be widely varied in accordance with the needs of a particular system. By way of example, typical tasks include flow classification, policy based routing, complex protocol and signature based filtering, encryption and decryption, and packet editing. Some of the most frequent uses of the traffic classifier are to identify routing destinations and determine routing priority for the incoming packets. Other tasks performed by the traffic classifier 204 may involve manipulating the packet. For example, the traffic classifier 204 may edit and/or filter the packet based on predetermined policies. In one specific example, the traffic classifier classifies the received packet to determines how the packet will be handled. Most commonly, the traffic classifier 204 determines at least a routing destination and/or a routing priority for the packet. The structure of one embodiment of the traffic classifier 204 will be described in more detail below with respect to Fig. 4 and is described in detail in Appendix A of U.S. Provisional Application No. 60/133,241, entitled Packet Processing Architecture and Methods, filed on 07 May 1999 by Deb et al, which appendix includes a document entitled "Functional specification of packet classifier chip" by Devendra Tripathi.

The information that the traffic classifier associates with the packet may be presented in a wide variety of forms. For example, the information may be summarized within a data manipulation layer (DML) header that is associated with the received packet as it is passed to the next component (e.g. the traffic manager). By way of specific examples, the header may be part of the packet (*e.g.*, attached or appended to the received packet), or the header may be received independently of the data packet (*e.g.*, the packet and header may be received separately on different buses or the header may be referenced by a pointer within the packet). The traffic manager may then use the traffic classifier information to perform its own tasks such as routing, prioritization and/or queuing.

The traffic manager 214 is arranged to route the received packets based, in part, on the information that is made available to it by the traffic classifier 204. Thus, the traffic manager facilitates policy based routing and scheduling. Structurally, the traffic manager includes a large number of queues and is arranged to manage these queues. Thus, one function is to determine which packets go into which queues. Other functions may include determining where to insert a packet within a particular queue, or what priority to give the packet for forwarding and/or dropping the packet and/or whether the packet requires specialized handling (*e.g.*, the packet is sent to an exception handler of the CPU). The queue determinations are typically made, based at least in part on the information supplied by the traffic classifier. In one preferred implementation, the traffic manager 214 uses the supplied traffic classifier information to quickly link the packet to a particular queue.

The monitor processor 216 is arranged to perform a variety of tasks including monitoring the device traffic and accumulating appropriate statistics. It may also perform functions that are similar to the tasks performed by the traffic classifier 204. For example, both the traffic classifier 204 and monitor processor 216 may include classifying, filtering, and data compaction tasks. However, the tasks of the monitor processor 216 and traffic classifier 204 may differ quantitatively. For example, the monitor processor 216 may be arranged to perform filtering tasks that are computationally more complex than the traffic classifier's 204 filtering tasks. The subtasks of the monitor processor 216 may also include tasks that differ qualitatively from the tasks of traffic classifier 204, such as flow based statistical database maintenance. As shown in Fig. 2A, a packet may be received in parallel by both the traffic classifier 204 and monitor processor 216 such that the tasks of the traffic classifier 204 may be performed in parallel with the tasks of the monitor processor 216.

As shown in Figure 2A, the traffic classifier 204 is arranged to receive incoming packets from an appropriate source. In the embodiment shown, the source of the incoming packets is a MAC device 202 which communicates with the traffic classifier through bus 224 and the monitor processor 216 through bus 226. The MAC device 202 may take any suitable form for communicating with a physical device. For example, the MAC device 202 may take the form of a XMAC device, available from XaQti Corporation of San Jose, California. Alternatively, the MAC device 202 may take the form of a state machine configured to receive the packets from a physical medium (not shown), perform functions such as, e.g., stripping the preamble (if present) from the incoming packet, and transmit the modified packets to the traffic classifier 204. Also, the MAC device 202 may be capable of passing CRC fields to traffic classifier 204 without performing a conventional stripping function.

The traffic classifier 204, the traffic manager 214, and/or the monitor processor 216 are programmable through buses 220 and 218. Preferably, the specific tasks performed by these blocks (*e.g.*, the traffic classifier 204, the traffic manager 214, and the monitor processor 216) may be programmed in parallel and/or while packets are still being received and/or transmitted. That is, while data packets are passed into the packet processing system 200 from either the MAC (through bus 224) or the switch fabric (through bus 214), control information designed to control the processing performed by the traffic classifier 204 and/or traffic manager 214 may also be simultaneously passed through bus 220. The use of parallel data and control buses provides the ability to change the processing parameters at any given time (*i.e.*, even while packet data is being processed) and therefore facilitates dynamic control of packet processing.

By way of example, suppose a packet is being received from the MAC device 202, and is being sequentially processed by various processing blocks. Due to the parallel processing nature of packet processing system 200, control information may be simultaneously passed through bus 220 to modify how the packet will be processed by processors that have not yet processed the packet . Accordingly, the parallel processing nature of packet processing system 200 is capable of passing appropriate control information to alter specific processing parameters of a particular processor even while the packet is currently being processed by another processor. In other words, processing of a particular packet may be dynamically changed on the fly as the packet moves through the packet processing unit 200. For example, the processing parameters of a first processor of the traffic classifier 204 may be altered while packets are being processed by a second processor of the traffic classifier 204. Thus, after the packet is processed by the second processor and when the packet reaches the first processor of the traffic classifier 204, the packet is then processed under the altered parameters. With this arrangement, modifications of processing parameters does not stop or significantly slow down the flow of the packets through the packet processing unit. Thus, the present invention facilitates high speed switching by providing an efficient mechanism for modifying packet processing without significantly reducing the flow speed of the packet through the packet processing unit.

In one embodiment, the processors include dedicated state machines for performing specific tasks that are programmable by a single instruction. In other words, a single instruction initiates a complex, multi step task This arrangement allows parallel performance of multiple tasks. The task may include, for example, switching a Layer 2 packet, routing a Layer 3 packet, performing a quality of service check to define traffic flows, performing a security check, or deriving accounting or billing information. Devices and techniques for programming the processors are further described below in reference to Fig. 5A.

The traffic classifier 204 interfaces with the monitor processor 216 through bus 218 and traffic manager 214 through bus 208. The traffic classifier is also arranged to interface with a memory device, such as an SRAM (not shown), through bus 206. The monitor processor 216 may be further arranged to interface with a memory device, such as an SRAM (not shown), through bus 222 and a CPU and/or other peripheral devices (not shown) through bus 220. The traffic manager 214 interfaces with a switching device (not shown) through bus 214, and may be further arranged to interface with a memory device (not shown), such as a SDRAM, through bus 212. Each of the above described buses may include a separate input and output bus or a single bi-directional bus.

Referring next to Fig. 2B an alternative embodiment of the packet processing system of the present invention will be described. Fig. 2B is an architectural diagram of a high speed packet processing system 280 in accordance with the alternative embodiment. In this alternative embodiment, in addition to the components described above with respect to Fig. 2A, the packet processing system 280 includes a stream interface synchronizer 262, a stream interface module 260, a traffic manager interface 252 and a traffic manager synchronizer 262.

The stream interface synchronizer 262 is arranged to match data rates between the MAC device 202 and the packet processing system 280. Thus, the stream interface synchronizer 262 converts the received rate of incoming data packets from the MAC to the internal processing rate of the receive side of the packet processing system 280. It also converts the internal transmission side processing rate to the transmission rate of the outgoing data packets that are transmitted to the MAC. For example, the stream interface synchronizer 262 may be arranged to convert a received rate of about 25 to 62 MHz to an internal rate of about 100 to 125 MHz. The received rate may be any frequency and depends on the type of MAC device used. Likewise, the internal rate may be any suitable rate for processing the packets without substantial congestion. In sum, the stream interface synchronizer 262 is arranged to receive data packets at a received rate and output the outgoing data packets at the internal data rate.

As will be appreciated by those skilled in the art, many state of the art MAC devices are capable of handling connections to a plurality of different ports. In such systems, the packets or packet segments that are received from the MAC may come from a plurality of different streams and there is generally no particular pattern that the packets/packet segments are received in. In implementations that are intended to handle packets from a plurality of different streams, the packet processing system 280 includes a stream interface module 260. The stream interface module is arranged to segregate incoming packet/packet segments into their associated streams and to accumulate packet segments from a particular media port sufficiently to permit processing by the traffic classifier. The packets may be segregated in a variety of way. In the described embodiment, the stream interface module 260 utilizes a plurality of linked lists to associate packets/packet segments with specific streams. That is, each stream has an associated linked list

To insure reliable processing/classification of the packets by the traffic classifier, it may be desirable to insure that a packet/packet segment has a length that is larger than a predefined threshold value. The predefined threshold may be set at any suitable value such that the processors of the packet processing system 280 will have enough information from the packet to reliably process it. After a large enough group of segment(s) are received, the stream interface module 260 is further arranged to transmit the received packet segment(s) to the traffic classifier 204 and/or monitor processor 216.

By way of example, as will be appreciated by those skilled in the art, one possible configuration of the MAC device 202 takes the form of three octet MAC devices that are configured together to receive packets from one of 24 different media ports. In such a system, a first packet segment may originate from port 1, while a second packet segment may originate from port 4 and a third packet segment originates from port 17. Segments are accumulated within an associated linked list (*e.g.*, one of 24 linked lists that are each associated with a media port) within the stream interface module 260. For example, the first packet segment is accumulated within the linked list associated with port 1, the second packet segment is accumulated within the linked list associated with port 4, and the third segment is accumulated within the linked list associated with port 17.

The lengths of the segments within each queue are periodically checked against the predefined threshold value to determine whether the packet is large enough for further processing by the traffic classifier 204. For example, the length of a particular linked list is checked each time a new segment is added to the particular linked list. By way of specific example, after the first packet is added to the first linked list, the length of the first linked list is checked and it is determined that it is not long enough. After the second packet is added to the second linked list, the length of the second linked list is checked and it is determined that the linked list or second segment is long enough for further processing. The stream interface module 260 then sends the second packet to the traffic classifier 204, while the first segment stays within its queue.

The packet processing system 280 may also include a local queue control block 254 arranged to control traffic congestion or provide various guarantees of service quality (QoS) during congestion, for example, in place of the traffic manager 214. The local queue control block 254 may take any suitable form for providing traffic control. For example, the local queue control block 254 may provide packet buffering to handle congestion. By way of another example, the local queue control block 254 may provide a plurality of queues, which may or may not have different priorities and destination ports for packet transmission.

The packet processing system 280 further includes a traffic manager interface synchronizer 250 arranged to match the traffic manager 214 data rate with the internal data rate and a traffic manager interface 252 arranged to act as a local switch. For example, the traffic manager interface 252 allows local traffic to be sent back to a local port via path 208b. This feature allows a data packet to bypass the switch fabric and be sent directly to a local port. As explained above, the MAC device(s) may be connected to a plurality of local ports. Data packets may be sent between one of these local ports and an external port via the switch fabric, or packets may be sent between two of these local ports through the traffic manager interface 252.

The traffic manager interface 252 may also select a packet source (e.g., from the traffic manager interface synchronizer 250, a special packet generator 256, or CPU interface 258) and/or check the parity of a transmitted packet. That is, the traffic manager interface 252 may receive either a packet from the special packet generator 256, CPU interface 258, or traffic manager interface synchronizer 250 and transmit the received packet to the traffic classifier 204 via path 208b. For example, the special packet generator 256 may be arranged to generate any suitable packet type, such as an error packet or a Internet Control Message Protocol (ICMP) packet, such as "time to live." Additionally, the traffic manager interface 252 may be arranged to copy a particular packet to the CPU interface 258, such as an error message, routing information, or unidentified packet.

The CPU interface 258 may be coupled to a CPU (not shown) that is associated with packet processing system 200. Other packet processing systems that are connected to the switch fabric 290 may also have an associated CPU. This configuration allows the CPU associated with a particular packet processing system to communicate and share information with other CPU's and their associated data processing systems through the switch fabric 290. In other words, this configuration represents an integral mechanism for using the switched data path to communicate with other CPU's without establishing a separate data path that couples the CPU's together. Additionally, such CPU communications may be assigned a particular bandwidth or a particular subdivision of a particular bandwidth based on any suitable criteria, such as priority).

The CPU interface 258 may be arranged to receive packets from the data stream and insert packets within the data stream, for example, via the traffic manager interface 252. For example, the CPU interface may read statistic counters generated by the monitor processor 216, or may read management information generated by individual processors within the packet processing unit. Additionally, the CPU interface may be used to perform various other programming functions, such as accessing and modifying routing information, programming and reading various registers within the packet processing unit, and/or reading available microcode from the packet processing unit.

The processors of the packet processing system 280 (*e.g.*, processors of the traffic classifier 204, monitor processor 216, and traffic manager 214) may be programmed in any suitable manner. As described above, the processors may be programmed by generating microcode that is input to the packet processing system 280 via a management bus. For example, programming data is written into hardware within the processors. Alternatively, the microcode may be input into a memory storage device 264, such as an EEPROM, flash EPROM, or through a general purpose serial port. Upon power-up, programming data is written into the processors' hardware.

Referring next to Fig. 3 a router formed using the described packet processing systems 200 or 280 in accordance with one embodiment of the present invention will be described. As seen therein, the router includes a plurality of packet processing systems 200 that are coupled together through a switching device 302. Each packet processing system 200 is coupled with an associated MAC device 310 through an associated bus 224, and each MAC device is coupled with an associated physical device 312. Each packet processing system 200 is further coupled to a plurality of memory devices 304.

A management processor system 314 is coupled to the packet processing systems 200, as well as the MAC devices, through management bus 220. The management processor system 314 is arranged to transmit control data through management bus 220 to the packet processing systems 200 and MAC devices 312. For example, the management processor system 314 may be used to select queuing priorities of certain data packets that are received into the packet processing systems 200. By way of another example, the management processor system 314 may be used to manage traffic load and/or prevent security violations. That is, individual packet processing systems 200 may be programmed to filter out packets from a particular source or to a particular destination. Alternatively, the packet processing systems 200 may be programmed to limit the percentage of bandwidth that is used by a particular group of users, *e.g.*, nonemployees vs. employees or for any policy based grouping of traffic.

Referring next to Fig. 4, one embodiment of the traffic classifier 204 of Figs. 2A and 2B will be described. Turning initially to the transmit side, a transmit bit stream processor (BSP) 414 and transmit BSP editor 416 are arranged to receive data from the switching device (not shown) through the traffic manager 210. The transmit BSP 414 and editor 416 are responsible for certain packet processing tasks, such as prioritizing the different types of data being transferred across a network (*e.g.*, audio, video, graphics, multicast traffic, etc.) into a number of linked buffers contained within a transmit FIFO 420 (or any suitable queue structure). Multicast packets may be replicated within the transmit FIFO for transmission over various ports. In this manner, traffic classifier 204 is capable of having multiple simultaneous streams of data flowing through transmit FIFO 420 at one time. Additionally the transmit BSP 414 may be arranged to check whether the packet is virtual local area network (VLAN) aware and perform VLAN filtering.

Turning to the receive side, the traffic classifier 204 includes a MAC bus interface 402 (which may include both the stream interface synchronizer 262 and stream interface 260 of Fig. 2B) for receiving data through bus 224a, a receive bit stream processor (BSP) 404 for parsing received data 428, a receive FIFO 406 (or any suitable queue structure) for holding received packets, a search engine 408 for determining how to handle the received packets, and a post search receive BSP 412 for further parsing and editing (*e.g.*, extracting information from the packet and modifying the packet with the extracted information or other information) or processing received data based at least in part on information provided by the search engine 408. BSP 412 may perform any number of edit tasks. For example, BSP 412 may flip bits, change bit positions, replace or cut portions from the packet, and/or append data to the packet. The parsed or edited packet is then transmitted through bus 208a to the traffic manager (*e.g.*, 210 of Figs. 2A and 2B).

The receive bit stream processors 404 and 412 are user programmable in-line packet processing engines that are capable of rapidly parsing through received packet data and/or associated data bases to build user defined data structures (e.g., header information) that may be associated with the received packet (*e.g.*, appended to the packet or transmitted in parallel with the packet) as the received packet is being transferred to the traffic manager 210. The receive bit stream processors 404 and 412 also are also capable of rapid, user programmable in-line editing.

To initialize a user programming operation of a particular processor, the user may configure a software instruction set designating the type of parsing to be performed on in-coming packet data, as well as the type of data structure to build and associate with respective packets or a sequence of editing actions that are to be performed on the packet through the use of a graphical user interface (GUI). Once the instruction set is configured, the software instruction set is compiled to executable "microcode" by the host's CPU, and then transferred into hardware memory locations resident in the integrated circuit core of the particular processor.

It is important to realize that packet data processing occurring in the particular processor rapidly generates the user defined data structures and/or modifies the packet in an in-line manner (i.e., without slowing down data transfer rates). Defined data structures are advantageously associated with the packets as they are streamed to upper processors. When the upper processors (e.g., search engine 408 and/or traffic manager 210) access the packet data and the associated data structures, the upper processors need not scan the entire packet byte-by-byte (or bit-by-bit) to identify packet data of interest. This is because the user defined data structure may be programmed to store pointers to portions within the packet data that may be of interest to upper processors, or portions of data (hashed or compressed data) that may be quickly read and processed without having to scan and process the entire packet. Once the upper processors receive the compactly coded data structure information, the upper processors may rapidly determine how to handle the packet data and stream the packet data to its intended destination (e.g., a switch, a router, a client, a server, etc.).

The receive bit stream processor 404 is arranged to generate a data structure (hereafter, referred to as a "key header"). To create the key header, the receive bit stream processor 404 parses through the received data packet and obtains information from the data packet itself. After the information is obtained from the data packet the extracted information may be added directly to the key header as is or the information may be added as a compressed or "hashed" form. Alternatively, the receive bit stream processors 404 may simply provide the key header with a pointer to the relevant information in the packet or even encode the information into bits or groups of bits. The bit stream processor 404 may also be responsible for writing the packet into the receive FIFO 406, while the search engine 408 is responsible for writing the DML header into the receive FIFO 406 The bit stream processor 404 may write the packet into the FIFO such that a space is reserved for the DML header.

Each packet or packet segment may require several associated key headers, key header portions, or "micro keys." For example, a key header or key header portion may be generated that includes forwarding information, and another key header or key header portion may include quality of service information. As the receive bit stream creates a key header or key header portion for a particular packet or packet segment, the key header or key header portion may be sent to the search engine 408. As other key headers or key header portions are generated for the packet or packet segment, it is also sent to the search engine 408.

Search engine 408 receives each key header through bus 434 and is arranged to use information in the key header(s) to quickly look up information relevant to the routing, prioritization, scheduling, queuing and/or further processing of the packet from one or more external databases. The search engine is also capable of dynamically maintaining state information by writing data into the look up database. For example, the search engine may track states, such as connection status, and add and/or modify state entries within the database(s), accordingly.

After looking up information, the search engine uses the looked-up information to create another data structure (hereafter, referred to as a "data manipulation layer (DML) header"). The DML header may be based on the key header and/or information obtained from one or more databases (not shown) through bus 206.

The DML header may include any suitable information that may be utilized by downstream processors. Typically, the DML header will identify a specific queue in traffic manager 214 that the packet is to be assigned to. In some cases, the queue assignment will inherently associate policy with the packet. For example, a data packet assigned to a first queue may have higher priority than a data packet assigned to a second queue. The DML header may also include a variety of other specific information. By way of example, it may indicate a priority that is to be associated with the packet. It may also include port identification, multicast or unicast information required by the switch fabric, policy information, queue priority, and/or security information that is associated with the packet destination . The DML may also include editing information for the post edit BSP 412, such as information that indicates whether to generate a new checksum and/or CRC or whether to replace the source.

After the search engine 408 creates the DML header based on the key header(s) and/or the database, it associate the DML header with the data packet. The DML header can be associated with the data packet in a variety of ways. For example, it may be inserted into an appropriate location within the receive FIFO 406. Alternatively, it may be output to the next component (e.g. post edit BSP 412 or traffic manager 210 in the event that no post edit BSP is provided) synchronously with the data packet or it may be placed in a parallel FIFO (not shown) that is synchronized with the FIFO 406 such that both FIFOs output information relative to the same packet at the appropriate time. In the described embodiment, the DML header is associated to the data packet within the FIFO 406. This DML header may then be used by the post edit BSP 412 to edit the packet itself and/or DML header and by the traffic manager 210 to determine how to route the data packet. Alternatively, the functions of the search engine 408 and

BSP engines 404 may be combined into a single block for creating DML header, or the search engine 408 and post search receive BSP 412 functions may be combined with the traffic manager 214 into a single block for managing queues based on the key header.

The search engine 408 may write the DML header using any suitable techniques. For example, the search engine 408 may determine when and where to insert the DML header into the FIFO based on how and when the packet moves through the FIFO. In one embodiment, the receive bit stream processor 404 writes the packet into the bottom portion of the FIFO after or concurrently with the key header being output to the search engine 408. As pointed out above when the receive bit stream processor 404 writes the data packet into the FIFO 406, it reserves sufficient room to hold the DML header at a specific location relative to the remainder of the packet. By way of example, the first FIFO cell that the packet occupies may be reserved for the DML header. As the packet moves up the FIFO at a predefined rate, the search engine 408 tracks this predefined rate by outputting DML headers at the same rate. When the packet reaches a designated location within the FIFO, the search engine 408 writes a DML header that is associated with the packet into the reserved location. By way of example, the designated location within the FIFO 406 may be a position where the packet is almost at the top of the FIFO such that the DML header is associated to the head of the packet.

Alternatively, the search engine 408 may write the DML header to the FIFO based on information supplied by the receive bit stream processor 404. In this alternative embodiment, the packet is written to the FIFO such that a space for the DML header is reserved. The bit stream processor 404 passes a pointer to this reserved space to the search engine 408 as part of the key header, and the search engine 408 uses the pointer to write the DML header to the reserved space in the FIFO.

The receive FIFO 406 may be any suitable size for holding the data while the search engine 408 generates the DML header. For example, the receive FIFO 406 may be sized to hold a maximum sized packet. A size of four kilobytes appears to work well. For example, a packet is held within the FIFO until an associated DML is associated with it. After the DML is inserted into the FIFO, the packet and DML header are transmitted to the post edit BSP 412 (if present) or traffic manager 214. In this embodiment, a larger FIFO may be located within the MAC device 202. Alternatively, the size of the receive FIFO 406 may be large enough to hold any number of packets at a given time. For example, the FIFO may accumulate a plurality of packets when traffic is particularly heavy. By way of another alternative, the receive FIFO may take the form of an external FIFO that is externally accessed by the traffic classifier 204. In other words, the traffic classifier 204 may be implemented on a single device or chip, and the traffic classifier 204 may access an external FIFO through pins of the device. More specifically, the bit stream processor 404 writes the packet to the external FIFO, and the search engine 408 writes the DML header to this external FIFO. The contents of the FIFO (*e.g.*, the packet and DML header) may then be read by the traffic manager 214.

As previously mentioned, each of the described components (*e.g.*, the traffic classifier 204, traffic manager 214, and monitor processor 216) may be programmable. That is the user may program bit stream processors 404 and/or 412 to generate custom data structures. By way of example, in one particular application, it may be useful to provide, a pointer to the start of an internet protocol (IP) header, a pointer to the start of a transmission control protocol (TCP) header, and a pointer to the start of a simple mail transfer protocol (SMTP) header. In another particular application it may be useful to provide the source network address, the destination network address, protocol type, destination port number, source port number to create a flow. Other exemplary data structures may be programmed to include portions of the packet data itself, such as compressed or hashed portions. Because the type of data structure and the content of the data structures are fully programmable by the user, other exemplary data structures may include "flag" data structures where each bit is programmed to identify one protocol or another or represent certain TCP "flag" bits or other bits of interest, or "field" data structures where multiple bits are coded to identify different networking protocols.

For example, the data structures or key header may include any suitable type of information that is relevant for further processing of the data. For example, the key header may indicate how to handle the packet, which database(s) to search, and what type of search algorithm to execute. The key information may further indicate a source and destination of the received data packet. By way of another example, the information may indicate the type of data within the received data packets. The data within the received packet may include payload data, as well as header data associated with the payload data. The key header may indicate whether the payload data is an audio, video, or data packet, whether the packet is part of a real-time or delayed session, whether the packet is associated with an existing session, whether the packet indicates an end of an existing session or a beginning of a session. Additionally, the key header may include security information, such as passwords, fields of interest to filters, encryption algorithm(s), and/or compression algorithm(s).

Any suitable programming apparatus may be implemented such that at least some of the subtasks of the processors are programmable. For example, the processors may include programmable registers that are accessible by a host CPU. One or more of the processors may be programmable. Although programming techniques and devices are described below in reference to Fig. 5A in terms of programming one of the bit stream processors 404 and 412, it should be well understood that the described devices and techniques may be modified and applied to programming any of the other processors, such as the search engine 408 or the traffic manager 214.

Figure 5A is a high level block diagram illustrating the interactions between a host CPU and one of the bit stream processors 404 and 412 of Fig. 4 in accordance with one embodiment of the present invention. As described above, because a user is able to program the type of processing to be performed in bit stream processors 404 and/or 412, the user is preferably directed to define a software instruction set identifying the type of data structure to build, as well as the content of the data structures. To achieve this, the user preferably implements a graphical user interface (GUI) 500 that provides a list of choices for ease of programming. The GUI 500 generates a software instruction set that is compiled by a CPU 501. The compiled software instruction set is then converted into executable microcode. In one embodiment, the microcode will preferably contain all of the user programmed information identifying the type of data structure to build, and the content of each data structure that will be associated with each packet received by bit stream processors 404 and/or 412. Alternatively, the user selects predefined keywords (*e.g.*, of a code generator) to generate a sequence of edits to be performed on a packet.

In another embodiment, a protocol descriptive language (PDL) may be used to define a class of protocols using a set of predefined mnemonics with well understood semantics. Example mnemonics may include IP v.4 address, protocol ID, 802.3, and SNAP-encapsulation. These mnemonics may also be combined to form a data structure. A PDL compiler program can then generate the microcode for bit stream processors 404 and/or 412.

In general, once the microcode for carrying out the processing within bit stream processors 404 and/or 412 has been executed by the CPU 501, the microcode is transferred to a bus 208/220, and then to a bus interface controller 504. The bus interface controller 504 then transfers the microcode to hardware storage locations within bit stream processors 404 and/or 412. Accordingly, the packet data may be simultaneously (*i.e.*, mirrored) or independently transferred through both or one of the network bus 208 and management bus 220. In one embodiment, a portion of the microcode is transferred into a random access memory (RAM) 502, a portion of the microcode is transferred into a content addressed memory (CAM) 534, comparators 536, and control registers 535. Once RAM 502, CAM 534 and comparators 536 have received the user programmed microcode, bit stream processors 404 and/or 412 will be initialized and ready to receive packet data from MAC device 202.

In this embodiment, RAM 502 is preferably a thirty-two bit wide (or wider) by 4096 deep static RAM, which contains an input register on the address input. Accordingly, when an address enable is high, the input register latches the address. Of course any other suitable storage device may be implemented, including a read only memory having pre-programmed microcode instructions or an FPGA. Further, CAM 534 preferably includes a set of sixteen 16-, 32-, or 48-bit registers with equality comparators. In this manner, data to be compared is latched into a register having an output that goes to each of the equality comparators, and flags from the comparator are added together to produce a match signal (matchfound). Still further, CAM 534 may contain a look up table corresponding to all the entries, and, when a match occurs, the corresponding entry is output. Control registers 535 may also be provided to store any suitable processing parameters, such as search, parsing, and editing parameters.

Any suitable programming techniques may be implemented. For example, as described above the microcode may be developed and communicated to the local CPU 501. The CPU 501 then writes the microcode to various memory. Alternatively, the CPU 501 may store various code segments for different policy implementations. As a particular policy is required or changed, the code segments associated with the new policy are then automatically downloaded by the CPU.

Figure 5B is an architectural diagram of an implementation of bit stream processors 404 and 412 of Fig. 4 in accordance with one embodiment of the present invention. Assuming that, RAM 502, CAM 534, and comparators 536 have already received the user-defined microcode from CPU 501 as described in Figure 5A, an initial portion of the microcode contained within RAM 502 is transferred to an instruction register 504. Further, the transferred microcode will preferably contain microcode information to set a word count 508. In this embodiment, the microcode that is resident in word count 508 is configured to identify a desired word count in an in-coming packet.

By way of example, each time a new packet is received by bit stream processor 404, a word counter 507 will reset to "0", and then word counter 507 begins sequentially counting each word that is received into pipeline register stages 523 from data path 403. As shown, pipeline register stages 523 preferably includes a "stage 1" 524, a "stage 2" 526, and a "stage 3" 528 that provides powerful processing capabilities. By way of example, if a stage 1 524 contains the 57th word, stage 2 526 will contain the 56th word, stage 3 528 will contain the 55th word, and a MUX 520 may select portions of the 55th, 56th and 57th word to process at one time. For ease of understanding, the advantages of the pipeline register stages 523 will be described in greater detail below. Of course, it should be understood that any number of pipeline stages may be used.

When a desired word count identified by word count 508 is received into pipeline register stages 523, the microcode initially stored in instruction register 504 will be transferred to an execution instruction register 506. As shown, a summing unit 505 is preferably configured to continuously receive the current word count number from word counter 507, which notifies word count 508 that it is time to transfer the microcode to the execution instruction register 506.

When this happens, the selected word of the in-coming packet has now been stored in stage 1 524 of the pipeline register stages 523. At the same time, the microcode contained within execution instruction register 506 is passed to execution logic 512 which controls the current action of bit stream processors 404 and/or 412. In one embodiment, execution logic 512 communicates with a MUX 520, a MUX 518, a MUX 514, a CRC unit 530, a HASH 531, an arithmetic logic unit (ALU) 532, CAM 534, comparators 536, and a programmable checksum generator 533. As shown, CRC 530, HASH 531, ALU 532, CAM 534, comparators 536, and a programmable checksum generator 533 are part of an analyzing computer 537 that is configured to act on the word of interest (of a current packet) identified by word count 508. As described above, if MAC device 202 passes the received packet along with the CRC field, CRC 530 is preferably configured to perform a CRC calculation and strip the CRC field before the packet is transferred to the upper LLC layer. In one embodiment, the CRC calculation is a 32 bit or 16 bit cyclic redundancy check using a generator polynomial.

Based on the execution commands provided by execution logic 512, execution logic 512 instantaneously programs the analyzing computer 537 as well as MUX 520 that selects the identified word stored in stage 1 524, or a portion of the words stored in stage 2 526 and stage 3 528. That is, if portions of words are selected from each stage to construct a new 32-word, MUX 520 will select that new 32-bit word and transfer it to a bus 540. Once the desired word has been transferred to bus 540, the analyzing computer that includes CRC 530, HASH 531, ALU 532, CAM 534, comparators 536, and programmable checksum generator 533 operate on the 32-bit word selected by MUX 520.

If the user desired to create a data structure having a pointer to the currently selected 32-bit word (which may be the start of a header), then the word count will be transferred from word counter 507 to MUX 518 to be input into a current data structure. In this embodiment, the current data structure will preferably be stored in a data structure register file 516. Once all of the data of interest for a current packet has been parsed and stored into the data structure register file 516, the data structure will be associated with the beginning of the packet data being output from MUX 514.

In another example, if the user desires to construct a data structure that includes hashed data (i.e., compressed packet data), the hashed data will be processed in HASH 531 and then passed to MUX 518. Still further, the user may desire that portions (*i.e.,* selected 32-bit words) of the received packet data be placed into the data structure for quick reference by upper layer protocols. Once an entry is made into the current data structure, for the current packet, CAM 534 and comparators 536 generate comparison control signals that are transferred to an encoder 517 and to a next address logic 510. The control information provided to encoder 517 is preferably used to set (*i.e.,* through encoded set bits) the type of data structure the user wants to build, and the control information provided to the next address logic is used to identify the next address from the microcode stored in RAM 502. Thus, based on the comparisons performed in CAM 534 and comparators 536, a branch operation or a move operation will be performed. By way of example, when a branch operation is performed, the next address logic 510 will locate another address location in RAM 502. Further, if a move operation is performed, one of the analyzing computer 537 units will transfer an output to MUX 518 and into data structure register file 516.

In one embodiment, the next address logic 510 contains logic for performing a vectored branch which identifies the next address in the microcode stored in RAM 502 based on the results obtained from the comparator's look up table contained within CAM 534. Further, conditional branching may be used to identify the next address from the microcode itself stored in RAM 502, based on outputs from comparators 537. Still further, Unconditional branch instructions may come directly from the microcode stored in RAM 502 without analyzing the comparison results generated in CAM 534.

As described above, once CAM 534 and comparators 536 of the analyzing computer 537 have examined the desired 32-bit word, the results of the comparisons are passed to the next address logic 510. In one embodiment, next address logic 510 will ascertain the next address location in the microcode stored in RAM 502 based on information provided from the execution instruction register 506 and the received comparison results received from CAM 534 and comparators 536. Meanwhile, each time a new address is located in RAM 502, that address is stored in a program counter (PC) 511. In this embodiment, the program counter PC 511 will keep track of the most recent address selected in RAM 502. Accordingly, program counter (PC) 511 is continually updated after each access operation into RAM 502.

Once the next location within the microcode contained within RAM 502 is ascertained, that portion of the microcode is again transferred to instruction register 504 and word count 508. Again, word count 508 will contain the next word count of interest within the current packet being received. By way of example, since the last word of interest was word 57, the next exemplary word of interest may be word 88. In this example, word 88 may identify the beginning of a header, the beginning of data to be compressed (*e.g.*, hashed) or the beginning of data to be captured. When word counter 507 reaches the 88th word in the packet, the microcode stored in instruction register 504 is shifted into execution register 506 to enable the executing on the newly received data word that is currently stored in stage 1 524 of the pipeline register stage 523.

Again, the contents of execution instruction register are transferred to execution logic 512 for programming the computation functions of the analyzing computer 537, and multiplexors 520, 514, and 518. As mentioned above, the data structure being built is preferably stored in data structure register file 516 before being passed to MUX 518. Once the entire data structure for a particular packet is stored in register file 516 (*i.e.*, after all of the positions within a current packet have been examined), the actual packet data that was being passed through pipeline register stages 523 is temporarily stored in a RAM FIFO 522. As controlled by execution logic 512, the user programmed data structure is passed into MUX 514 where it is associated with the packet data being received from RAM FIFO 522.

MUX 514 then outputs the packet and associated data structure to the multi-packet queue FIFO 406 as described above with reference to Fig. 4. Once the processing is complete for one packet, the next packet is again analyzed based on the same microcode provided by the user. However, if the user wants to modify the processing set in the software instruction set input through GUI 500 of Figure 5A, the received packets will be processed in accordance with those new parameters. As noted earlier, the data structures created for each packet may include only pointers to selected locations in the packet, only portions of data from the packet itself, only hashed data from the packet itself, or a combination thereof. Accordingly, bit stream processor 404 and/or 412 will operate on different packets in accordance with the specific microcode programmed by the user. While such a described architecture is believed to work particularly well, it should be appreciated that similar functionalities can be accomplished using other architectures as well.

When the upper processors (*e.g.*, search engine 408 and/or traffic manager 210) receive the packets having the associated data structures, the upper processors may simply read the data they need to complete packet routing without having to examine substantially all of the received packet to locate the information that is of interest to the upper layers. It should be noted that most of the time, each packet may have similar header information (i.e., the IP header) located in different byte locations within a received packet, and therefore, a host's CPU is typically required to laboriously scan through most of the contents of each packet before it can perform any necessary routing or processing. Accordingly, by associating a user defined data structure (*e.g.*, it may be sent in parallel with the received packet, or it may be appended to the front of the received packet), even greater than gigabit Ethernet or SONET transmission speeds may be attained with substantially fewer CPU interrupts.

Figure 6 is an overview flowchart diagram of the operations performed within bit stream processors 404 and/or 412 of Fig. 4 in accordance with one embodiment of the present invention. The method begins at a step 602 where a user defined software instruction set is loaded into bit stream processors 404 and/or 412 for programming the processing performed on packet data being received from MAC device 202. By way of example, the software instruction set is preferably programmed into a host receiving computer through the use of a graphical user interface (GUI) which prompts the user to define the desired processing on the received packets.

Accordingly, once the user has defined the type of data structures and the word count positions of interest within the received packets for which a data structure will be constructed from, the software instructions set is compiled. In general, the software instructions set is compiled into executable microcode which is then loaded into the RAM 502, CAM 534 and Comparators 536 as described with references to Figure 5B above. Once bit stream processor 404 and/or 412 has received the desired microcode that dictates the type of processing performed on the received packets, the method will proceed to a step 604 where an initial skip will be performed within the received packet. That is, once the packet is received by bit stream processor 404 and/or 412, an initial skip into the packet will be performed to determine what type of packet has been received. By way of example, such packets may include proprietary tagged packets, or any other type of packet that may be defined in the future. Generally, the initial skip is ascertained from the MAC layer protocol. Therefore, for Ethernet, the initial skip may have a skipping length of about 12 bytes. Of course, if other protocols are used, such as, for example, Token Ring or FDDI, other initial skipping lengths may be implemented as well.

Once the initial skip has been performed into the received packet, the method will proceed to a step 606 where the received packet is examined by an analyzing computer contained in bit stream processor 404 and/or 412 in accordance with the user defined processing instructions provided in the user defined microcode. By way of example, the packet is typically examined to ascertain the word count location of a particular header, or a particular piece of data. Once the analyzing computer computes the desired data to be associated with a data structure, the desired data is passed to a multiplexor.

After the packet has been examined in step 606, the method will proceed to step 608 where the identified pointer, data, or hashed data (or a combination thereof) is stored into the defined data structure. The method will then proceed to a decision step 610 where it is determined if there are any more positions of interest in the examined packet. By way of example, if the microcode determines that there are five separate headers locations of interest, then the method will proceed to a step 612 where bit stream processor 404 and/or 412 will skip to new position in the received packet in response to the examination of the received packet. In this embodiment, the new skip location will preferably be the location ascertained by the microcode address selected by a next address logic unit contained within bit stream processor 404 and/or 412.

In one embodiment, multi-gigabit speed Ethernet or SONET transmissions are contemplated (*e.g.*, 2.5 gigabits per second and higher). However, it should be appreciated that the architecture is equally applicable to other transmission protocols (*e.g.*, native IP, ATM, FDDI, and Fiber Channel) and both higher and lower speed transmissions. Additionally, although the present invention is described in terms of the Open System Interconnection (OSI) layer model, the invention may be implemented to work with other layer models.

Referring next to Fig. 7, one embodiment of the search engine 408 of Fig. 4 will be described. As shown, the key header is input to the search engine 408 through bus 434 and used by the search engine 408 to generate the DML header which is output through bus 436. The search engine 408 performs a number of functions based, in part, on the received key header. For example, the search engine 408 performs one or more searches on one or more databases based on information provided within the key header. The key header indicates how to search and for which address entry to search and in which database(s) to search. The search engine 408 uses the search results (*e.g.*, information obtained from the database(s) during a search) and/or information from the key header to create the DML header. Additionally, the search engine 408 may implement learning functions for making additions to the database(s) based on key headers that have no corresponding address entry and aging functions for purging unused entries from the database(s).

The search engine 408 may be configured in any form that is suitable for searching one or more databases based on a received key header. One possible configuration of the search engine 408 is represented in Fig. 7. In this embodiment, the search engine 408 is arranged to perform various predefined functions, such as a plurality of predefined searching algorithms, learning, aging, billing, and accounting functions.

These predefined functions require minimal CPU programming. In other words, these functions may be launched by a single corresponding key header executed by a single microcode instruction. The search engine 408 is also arranged to perform custom user defined functions. Thus, the present invention provides an efficient and flexible mechanism for processing packets.

As shown, the search engine 408 includes an input scheduler 702 for receiving key headers or key portions through bus 434. The input scheduler 702 also checks an entry cache 708 for previously input key headers or key portions. The entry cache 708 contains previously obtained search results for previously input key headers or key portions. The size of the entry cache 708 can be widely varied to meet the needs of a particular system. For example, the entry cache 708 may hold one entry per stream. of previously input key headers or key portions and their associated search results. In other words, a key header or key portions that is identical to the current key header or key portions may have already been received by the search engine 408 and stored within cache 708, along with associated DML information. Thus, frequently used DML header information may be quickly obtained and used to generate the DML header without searching through the databases. The input scheduler 702 may also coordinate invalidation of entries within the entry cache 708, for example, when the main search database is modified.

The input scheduler 702 may also have storage capabilities for accumulating a plurality of key headers or key portions that are required for a particular packet or packet segment. That is, a particular packet or packet segment may have a plurality of associated key headers or "micro keys," and each micro key may be used to initiate a specific search engine 408 task. In one embodiment, the micro keys are accumulated and ordered such that the search engine 408 may perform a set of tasks in a particular order. The order may be any suitable order and depends, for example, on the requirements of the particular application. For example, a forwarding key may be placed before quality of service and learning keys. This key buffering feature allows some elasticity in key processing by the search engine 408. Alternatively, some of the key portions or micro keys are accumulated until all the micro keys are present for a specific task, such as forwarding. The micros keys are then output to one or more search engines 704.

The engines 704 receive the micro key(s) and performs any number of functions based on information within the key header. By way of example, one engine 704 may perform a search for a specific data entry within one or more databases based on key header information. The engines 704 may also be capable of performing a plurality of different search algorithms that may be selected based on key header information. Several search algorithms and search data structures are described in detail in Appendix A of U.S. Provisional Application No. 60/133,241, entitled Packet Processing Architecture and Methods, filed on 07 May 1999 by Deb et al, which appendix includes a document entitled "Hardware Based Search Algorithms of TeraPower-

CL™" by Devendra Tripathi.

As described above, the key header may include information that facilitates packet processing and routing functions. For example, one key header may include port information, while another key header includes security information for a particular packet. Other key headers may include quality of service and/or routing information. One embodiment of a key header is described below in reference to Table 1.

The search engine 408 includes an address module 710 that generates an address bus based on address control signals that are output by the engines 704. The address bus are used to access a memory device (not shown). The memory device may contain one or more databases that are accessed by the engines 704. Alternatively, there may be more than one memory devices with each containing one or more databases that are accessible by the engines 704. The address signals are used to access a specific address entry within the database(s) that may be associated with a data entry or data pointer. In other words, each accessible address of the memory device may include both an address entry and an associated data pointer.

The engines 704 are configured to search through a plurality of address entries by varying the address signals. The engines 704 specifically search for a particular address entry that corresponds to a portion of the received key header. When a corresponding address entry is found, an associated data entry may be retrieved (if one exists for the corresponding address entry) via a data pointer that is associated with the corresponding address entry. Alternatively, the data entry may be directly associated with the corresponding address entry without using a data pointer. The retrieved data entry may then be used to generate at least a part of the DML header, which is output from the search engine 408 via the output scheduler 722.

The search engines 408 also include a data module 712 that is used to read data (*e.g.*, a data pointer or data entry) from the database(s). For example, when the engines 704 output a specific set of address signals to the address module 710, an associated data pointer value is output from one of the database(s) of the memory device to the data module 712. The engines 704 may then change the address signals to the address module 710 to correspond to the data pointer and access the data entry from the data module 712.

The data module 712 may also be configured to perform a write to one of the databases. The write may be initiated in any number of circumstances. For example, a write operation may be performed during a learning function, wherein new data entries are added to the database(s). By way of another example, a write operation may be performed to "age" the data entries, wherein each time a address entry is accessed (or "hit") it is marked to indicate use. The address entries may then be periodically checked for marks, and address entries and associated data pointers and data entries that are not used for a predefined period of time may then be purged from the database(s). The learning and aging functions are further described below.

The search engines 408 may also include a memory controller 714 that is used to generate required control signals for accessing the memory device. For example, the memory signals may include a chip select signal to enable the memory device, a read and write enable signal(s) to enable a read or a write to the memory device, and one or more byte enable signals to enable access to specific bytes of data within the database(s). In sum, the memory controller 714 provides control signals to the memory device; the address module 710 provides address signals to the address bus of the memory device; and the data module 712 provides data signals to the data bus of the memory device and reads data signals from the memory device.

As described above, the search engines 408 may also include learning capabilities for writing new data entries to one or more database(s). The learning function may be implemented by one of the engines 704 or by the CPU through an external access port block 716. Specifically, the CPU may write a new data entry to the memory device through the external access port block 716 and data module 712.

The corresponding address entry and data pointer may also be written to the memory device. The address entry may be specified by the input scheduler 702 via a received key header or by the CPU itself. In the later case, CPU is informed of relevant key by the search engines 408. Any suitable mechanism may be used by the search engine 408 to inform the CPU, such as an interrupt.

By way of a specific example, a key header may include source address information, and the corresponding port information is known by the MAC interface 402 of the packet processing unit. For instance, the MAC interface 402 may know the port information for all local ports that are connected to the MAC device. Thus, when a new connections is established on a particular port, a data entry may be created in the databases that includes the port information and is associated with the corresponding address entry (*e.g.*, the address of the source). Thus, the learned port information may be retrieved from the database for a next key header. In other words, the key field of the next key header may contain the previously learned address. Thus, when the search engine 408 searches for the key field of the next key header, the learned address and associated port information may be found within the database.

The search engines 408 may also include an aging controller 720 that deletes inactive address entries and associated data pointers and data entries from the database(s). That is, data entries are purged if they are not used by the engines 704 to generate the DML header after a predefined period of time. The aging controller 720 works with the input scheduler 702 and engines 704 to mark data entries within the database(s) that are found using the received key header. The aging controller 720 includes a timer that is used to periodically clear the marks from the data entries. The data entries may then be marked again when accessed by the engines 704. After the predefined time period, the data entries are checked to determine whether any entries are obsolete. In other words, if a particular data entry has not been marked within the predefined time period, it is purged from the data base by the engine 706 via the data module 712. Thus, data entries that are not used or accessed for long periods of time are deleted from the database(s).

The aging features allows the search engine 408 to conserve memory space that is taken up by inactive database entries. That is, data entries that are no longer needed or accessed are purged from the database. Additionally, this feature facilitates faster look-up speeds by minimizing the search area within the database that may be traversed by the engine 706 during a search procedure. For example, an infrequent user may initiate a session to a particular port, transfer data between itself and the port, and then close the session. After the session is closed, the infrequent user may not access the port again for a relatively long period of time, if at all. Thus, data entries that were entered into the databases for the infrequent user (*e.g.*, through learning) may not be needed again for a long time, if at all. To more efficiently use the database, entries for the infrequent user are deleted from the database by the aging controller 720.

The search engines 408 may also include register resources 718 that may be used for any suitable storage purposes. For example, one or more of the registers may be programmable and used to alter search algorithms. Additionally, some of the registers may be used as intermediate storage registers to temporarily hold information obtained from the database(s). The stored information is released when the engines 704 complete its searches for a particular set of keys and then used to create a DML header.

The completed DML header is output through an output scheduler 722 to the receive FIFO 406 (see Fig. 4). The output scheduler 722 may also include storage capabilities for accumulating DML information until the engines 704 complete its searches for a particular set of keys. The output scheduler 722 may also include a timer for determining when to output the DML header to the receive FIFO 406 such that the DML header is associated with the packet. In other words, as discussed above, the output scheduler 722 may wait to write the DML header until the packet is positioned at a designated location within the FIFO.

One embodiment of a key header will be described referring to Table 1. As shown below in Table 1, the key header is divided into a plurality of fields. Of course, it should be well understood that other fields may also be included within the key header and that all of the fields of Table 1 are not required to practice the present invention. In other words, the key header may include any suitable fields that may be used by the engines 704, for example, to perform a search or by other processors of the packet processing unit, such as the monitor processor 216 (Fig. 2A) or post edit bit stream processor 412 (Fig. 4).

Field Name	Number of Bits
dataSize	5 bits
streamNum	5 bits
action	3 bit
type	2 bits
firstKey	1 bit
lastKey	1 bit
subType	2 bits
keySize	5 bits
keyInstruction	2 bits
keyParam	16 bits
key	(depends on keySize)
editData	(remaining dataSize)

Table 1

Although each of the fields are listed above as having a predefined size (*e.g.*, bit size), of course, the given sizes are not meant to limit the scope of the invention, and the size of each field, as well as the specific fields provided may vary.

Each field specifies a type of information that may be used by the search engine 408 to perform various packet processing tasks, such as a database search for forwarding information. For example, the dataSize field specifies the length of the entire key header in number of words. In this embodiment, the key header size may have a size that is in the range of about 2 to 32 bytes of data. The search engine 408 uses the specified length to determine where to extract information from the key header that may be used to conduct a database search, for example. That is, the search engine 408 may extract all needed information from the start of the key header to the specified length of the key header. However, the dataSize field is not necessary if the length is known, *e.g.*, the length of the required information is predefined or fixed.

The streamNum identifies the stream. By way of example, the MAC device 202 may be in communication with 32 different local ports. Thus, the incoming packets may originate from any one of these 32 ports. For example, a first stream may originate from port 2, and a second stream may originate from port 32. The search engine 408 requires knowledge of the stream's origin. This information is supplied within the streamNum field. In some circumstances, the streamNum may correspond to the stream's port number.

The action field identifies what general action will be undertaken by the search engine 408. The action field may include any suitable predefined actions, such as forwarding, learning, aging, quality of service, security checks, accounting, and specified database accessing. Alternatively, the action field may indicate a user defined action or custom action that is to be performed by the search engine 408.

Each packet may require a number of actions. Thus, the key header for each packet will likely have a plurality of associated fields or micro key headers, and each key header indicates and facilitates a particular action. For example, a first key header may indicate a forwarding action, and a second key header may indicate a security action. Both actions are to be performed for a particular packet or packet segment.

Generally, the search engine 408 performs a forwarding action by determining where to send the packet. For example the search engine 408 finds destination port information for the packet from the key header and/or database(s). Learning is performed by determining whether the key header is new and does not have an associated address entry and data entry within the database. If the received key header is new and unknown, a new data entry may then be created, for example, via the CPU for the new key header. Aging is performed by determining which data entries are inactive for some predefined amount of time so that the inactive data entries may be deleted. The aging action field may be internally generated by a request from the aging controller 720.

The search engine 408 perform a quality of service action by finding priority information from the key header and/or associated database(s). Priority may be determined within a service context, as well as a routing context. In a routing context, quality of service may be based on the flow of packets. For example, during heavy congestion, certain packets may be dropped, while other cannot. In a service context, certain packets may be held up within a queue, while other packets must be forwarded almost immediately to their destinations. By way of specific examples, audio packets that form part of a telephone conference should be consistently and timely transmitted, while packets that are part of a file downloading procedure may be transmitted intermittently.

The search engine 408 may perform a security check by determining whether a particular source is blocked from accessing a particular destination. Additionally, the search engine 408 may authenticate a packet's password or encryption algorithm. For example, the keyParam field may include compressed or hashed data that may be used to retrieve the authentication entry from the selected database. The search engine 408 searches for an address entry that corresponds to the keyParam within the database and obtains corresponding security information from the database. The search engine 408 may then use this security information to authorize the packet.

This feature provides an effective security mechanism since the search engine 408 is capable of checking the security of every packet that belongs to a particular session. That is, each packet may contain security information that may be checked by the search engine 408. Alternatively, the search engine 408 may be configured to spot check or periodically check certain packets within a session.

The account action includes collecting account and/or billing information from the key header and database(s). This accounting information may then be passed to the monitor processor 216, for example, for further processing. The keyParam field may be utilized to store relevant accounting data (*e.g.,* in a hashed format). The accessing action may be used to initiate communication with the memory device. In other words, this action specifies that a read or a write is to be performed on the memory device.

Referring back to Table 1, the key header may include a type and subtype to define the search to be performed by the search engine 408. Of course, any number of fields may be used to define the search, such as a single field or multiple fields. In this embodiment, the type field is used to indicate a type of search algorithm, such as a direct look-up or linked list search, and the subtype field indicates which database to search. For example, the type and subtype fields may together indicate which database to search, and how to search it. Each database is constructed for a particular search algorithm. However, since more than one database may be used for each search algorithm, the subtype is used to specify the database Several types of search algorithms and database formats are described further below.

The firstKey field indicates whether the key header is the first key for a particular packet or packet segment, and the lastKey indicates whether the key header is a last key. These fields may be useful when a packet has multiple key headers. For example, the input scheduler 702 may store key headers for a particular packet until the lastKey field is set, and then the stored key headers and last key header are sent to the engine 702.

The keySize indicates the size of the key data that is located within the key field. The key data may consist of any relevant information that facilitates search engine 408 tasks. For example, the key data will typically consist of destination information. The search engine 408 may use the destination information to find associated forwarding, quality of service, quality of routing, and security information. The keyInstruction field indicates whether the packet is to be sent to the CPU or discarded. If the packet is to be sent to the CPU, this field also indicates whether the packet is sent after a search is performed and a new DML header is generated or sent with the same key header.

The key data may be used to search the database(s). That is, the engine 702 searches for an address entry that corresponds to the key data within the database. Of course, as described above the engine 702 may also search for the keyParam field within the database. The remaining key header is used as a editData field and includes packet editing information that may be used by the post edit bit stream processor 412 to modify the packet.

The search engine 408 may be configured to perform any suitable type of search, such as a user defined search or a predefined search. Each database may take any suitable form that facilitates one or more types of predefined searches. For example, the search engine 408 may perform a direct look-up search. The search engine 408 searches for the key field within one or more selected databases. That is, the engines 704 search through the selected database for an address entry that corresponds with the key field data. When and if a corresponding address entry is found, the data entry (if it exists) may be used to generate at least a portion of the DML header. It is possible that the outcome of the search may trigger another search before final data is found.

Another search type is referred to as a linked list search. When this type of search is specified by the key header, the search engine 408 searches through a selected database having a linked list format. In general, the search engine 408 uses a portion of the key header (*e.g.*, the keyParam) to obtain a pointer within a linked list database. The pointer indicates where to start searching within the database. For example, the engines 704 sequentially search through a group of address entries that are linked together by link pointers.

One embodiment of a linked list database 900 is illustrated in Fig. 8. As shown, the linked list database 900 includes a plurality of address groups 904 that may be accessed by a selected anchor pointer 910, which is specified by a portion of the key header (*e.g.*, the keyParam field). The selected anchor pointer is located within a plurality of anchor pointers 908 within a database. A first address group 904a includes four rows of address entries 902a through 902d. Each row contains a plurality of address entries 906 with each address entry being associated with a pointer to a data entry or with the data entry itself. A second address group 904b is linked to the first address group via link 912. Any number of address groups may be chained together. There may also be multiple links (*e.g.*, an UP and a DOWN pointer) that can be chosen based on some criteria (*e.g.*, whether the key value is greater or less than the address group's entries).

When a linked list search is selected, an anchor pointer (e.g., 510a) is retrieved from the anchor pointers 608 based on the keyParam field, for example, of the key header. The retrieved anchor pointer is associated with a particular address group (e.g., group 604a). The search engine 408 then searches through the particular address group to find an address entry that matches the key data. When a matching address entry is found, the associated data entry for the matching address entry is retrieved via the associated data entry pointer. When the end of a particular address group is reached (*e.g.*, when a match is not found within the first address group), the link 912 may be used to search the next address group.

The linked list structure provides an efficient search mechanism. That is, the specified anchor pointer allows the search engine 408 to begin the search within a particular address group, as compared with searching sequentially through all address entries in a non-hashed search. Preferably, two links (*e.g.*, a link to a higher address group and a link to a lower address group) may be provided with each address group. When the search engine 408 reaches the bottom of such an address group, it determines whether the key data is within a lower or higher address group and follows the appropriate link to continue the search within a lower or higher address group. This search mechanism may significantly decrease the search time. In particular, the search time becomes $\log_2 n$, where n is a time period that is required by a non-branching search algorithm.

Another type of search that may be specified by the key header is a maximum match type search. A key portion of the key header is divided into a plurality of parts. The key portion may be divided into any number of parts having variable or fixed lengths. Each part of the key portion is then compared with address entries within the selected database. When an address entry is found that matches a predefined minimum number of key parts, it is determined whether a valid data entry exists for the matched address entry. If a valid data entry is found, the search is stored. The search continues until a higher number of key parts are matched to a particular address entry. Once the complete key has been processed or no more matches are found, the previously stored pointer is used to get the data entry. If there is no previously stored pointer, it implies that an entry does not exist.

Each of the above described search algorithms may be implemented on any suitably formatted databases. For example, the direct look-up search may be implemented within a tag based, VLAN (virtual local area network), or flow database. The tag based database may be used for accelerated routing, and the VLAN database may contain map information for eligible target ports. The flow database may be used for quality of service and quality of routing data.

By way of another example, a linked list search may be implemented within MAC address, IP multicast, flow, or authorization databases. The MAC address database is used for layer 2 forwarding and learning, and the IP multicast database is used for IP multicast routing. The flow database may be used for quality of service and quality of routing data, and the authorization database is used for security checks.

By way of a final example, a maximum match search may be implemented within an IP routing database that has fixed match sizes or variable match sizes. A variable match size database 800 will be described in reference to Fig. 9. As shown, the database includes a plurality of levels. To clarify discussion, the database includes an upper level 802a and a lower level having element 802b and 802c. Of course, a database may include any number of levels.

The search is initiated at the upper level element 802a, for example. In this embodiment, the upper level element 802a, as well as the lower level elements 802b and 802c, include a plurality of fields. In this embodiment, each particular level element 802 includes a next entry pointer that links the particular level to a lower level (*e.g.*, level 802a is linked to level elements 802b and 802c via pointer 804a). Each particular level element also includes a match entry flag

806 that indicates whether the particular level element contains a valid data entry. For example, this flag may store the data entry pointer if a match at this level occurs.

The search process determines whether a match exists. The next match length 810 indicates the length of the pattern to be matched during the search comparison of the next level. The key data from the key header is matched with the match pattern 812. If a match occurs, a corresponding data entry may be obtained from an entry data pointer 814 (*e.g.*, if there is a valid data entry).

As described above, if a match is found, the data entry is stored and the search continues to the next lower level or next element of the current level. For example, if a match is found at element 802a, a data entry may be obtained from entry data pointer 814a and stored, while the search proceeds to the next element 802b of the lower level. If a match is found for the next element 802b, the search continues to the next element 802c of the same lower level. When a match is not found in any element (*e.g.*, within element 802c), the search is complete and the data entry is used that was matched within the lowest level data entry. In this example, the data entry obtained via entry data pointer 814b is used since the element 802b contained the last match.

The number of searchable elements that are within the lower level may be determined from a next match record number 806 of the preceding level. As shown, the next match record number 806a of the upper level element 802a indicates that the next level contain two elements 802b and 802c. The next match record number 806 may be used after a search of a particular level is complete and a match has been found, for example.

The search algorithms may be implemented differently based on which database is selected. For example, in the context of a linked list search, the packet is processed based on whether search results are obtained and which database is searched. When a MAC database is used and a MAC address is not found, received packet or packet segment is flooded. In contrast, when an IP multicast database is used and search results are not obtained, the packet is dropped.

As described above, a DML header may be output from the search engine 408 through the output scheduler 722. The DML header is based on one or more key headers that are associated with a particular packet or packet segment. One embodiment of a DML header having a plurality of fields is summarized below within Table 2. Of course, it should be well understood that other fields may also be included within the DML header and that all of the fields of Table 2 are not required to practice the present invention. In other words, the DML header may include any suitable fields that may be used by other processors, such as the monitor processor 216 (Fig. 2A), post edit bit stream processor 412 (Fig. 4), or traffic manager 214 (Fig. 2A).

Field Name	Number of Bits
dataSize	6 bits
streamNum	5 bits
discardPkt	1 bit
snoopPkt	1 bit
multiCast	1 bit
editFlag	1 bit
useQueueID	1 bit
portNumMaskSize	8 bits
priority	8 bits
mask	
queueID	16 bits
editInst	16 bits
editData	n words (depends on editInst)
customData	(depends on dataSize)

Table 2

Although each of the fields are listed above as having a predefined size (*e.g.*, bit size), of course, the given sizes are not meant to limit the scope of the invention, and the size of each field may vary.

Each field specifies a type of information that may be used by other processors to perform various packet processing tasks, such as forwarding. The fields may be predefined fields that are handled by the other processors in a particular way. Alternatively, a field may contain custom data, which may include any information.

The predefined fields include information that is used by other processors to perform predefined tasks. For example, the dataSize field specifies the length of the entire DML header in word units. In this embodiment, the DML header size may have a size that is in the range of about 2 bits to 64 kilobytes of data. The traffic manager 214 uses the specified length to determine where to extract information from the DML header that may be used to select a queue for the packet. That is, the traffic manager 214 may extract all needed information from the start of the DML header to the specified length of the DML header. However, the dataSize field is not necessary if the length is known, *e.g.,* the length of the required information is predefined or fixed.

The streamNum field of the DML header is passed through from the key header streamNum field. The discardPkt field indicates whether to discard the packet. The snoopPkt field indicates whether to snoop the packet. "Snooping" is a security check that includes verifying whether the packet is authorized to go to the specified destination. The multiCast field indicates whether the packet is transmitted in a multicast mode. That is, is the packet sent transmitted to a plurality of destinations that belong to the same group as the source.

The editFlag indicates whether the packet requires editing by the post edit receive BSP 412 (Fig. 4). If this field is set to indicate that editing is required, the editInst and editData fields become relevant. The editInst field may contain an editing instruction that is implemented by the post edit receive BSP 412, which will interpret the instruction and edit the packet accordingly. By way of specific examples, the editInst may include an instruction to calculate a new checksum or CRC field for the packet, cut the CRC field from the packet, or perform a MAC, IP, VLAN or custom edit. A MAC and IP instruction may include replacing the source and/or destination; the IP edit may also include cutting the destination; and the VLAN edit may include replacing or cutting the VLAN. The editData field may contain editing data that may be used by the BSP 412 to carry out the editing instruction. For example, the editData field may contain a next hop address that will be used to replace the destination address.

The portNumMaskSize field is used to indicate the port number when multicast is not selected. In contrast, when multicast is selected, the portNumMaskSize field indicates the size of a mask that is used to select ports for multicast transmission. The mask field would then indicate which ports to mask. In other words, each bit of the mask field corresponds to a port. When a bit is set to a logical 1, the corresponding port is masked and cannot receive the packet. When the bit is set to a logical 0, the corresponding port is not masked and is able to receive the packet.

The useQueueID field indicates whether a queue has been defined within the queueID field of the DML header. The traffic manager 214 may use the queueID to select the queue that will receive the packet. Usually, the useQueueID field will not be set, and the traffic manager 214 will select a queue based on port number and priority of the packet. More specifically, the priority field contains quality of service or routing information for the associated packet that may be used by the traffic manager 214 to select an appropriate queue for the packet and determine when to transmit the packet from the queue. The priority field may also be used to determine whether to drop the packet. As described above, the port is specified by the mask and portNumMaskSize fields.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. For example, the packet processing system may include a traffic classifier and a traffic manager, but need not include a monitor processor. By way of another example, the receive FIFO 406 of the traffic classifier 204 may be located within the receive bit stream processor 404 such that data is received by the receive bit stream processor 404 and parsed as the data moves through the FIFO. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

CLAIMS

1. A packet processing system for receiving and processing a data packet, the processing including a plurality of tasks and the packet processing system being coupled with a switching device for transmitting the packet after it is processed by the packet processing system to another network device, the packet processing system comprising:

a first processor arranged to perform a first subset of tasks on the data packet and associate a first header with the packet, the first header being based on information obtained while performing the first subset of tasks;

a second processor arranged to perform a second subset of tasks for the packet based on the first header supplied by the first processor; and

wherein the first subset of tasks differ from the second subset of tasks.

2 A packet processing system as recited in claim 1 wherein the first and second processors perform their associated tasks on the packet in a pipeline manner.

3 A packet processing system as recited in claim 1 or 2 wherein the first processor receives the packet from a media access control device.

4. A packet processing system as recited in any of claims 1-3, wherein at least one of the first and second processors is programmable.

5. A packet processing system as recited in claim 4, wherein at least one of the first and second processors includes a memory device arranged to receive a programming instruction that controls at least a portion of the tasks of the at least one of the first and second processors.

6 A packet processing system as recited in claim 4 or 5, wherein the first processors is programmable such that a custom first data structure is created and associated with the data packet.

7. A packet processing system as recited in any of claims 1-6, wherein at one of the

first and second processors is programmable while data is passing through the packet processing system.

8. A packet processing system as recited in any of claims 1-7, wherein the first and second processors are capable of processing packets at about a 2.5 gigabit per second speed.

9. A packet processing system as recited in any of claims 1-8, wherein at least one of the first and second processors are configured to access an associated database contains information associated with the packet.

10. A packet processing system as recited in any of claims 1-9, further comprising:

a management bus for programming one or more of the first and second processors to modify one or more tasks of the first and second subset of tasks; and

a network bus for receiving the packet.

11. A packet processing system as recited in claim 1, wherein the packet processing system may form part of a device selected from a group consisting of a firewall/intrusion detection device, a load balancing cluster switch, a powerful RMON, a directory enabled and content aware networking device, and an internet server provider (ISP) class edge router.

12. A packet processing system as recited in any of claims 1-11, further comprising a second processor arranged to perform a third subset of tasks in parallel to at least one of the first and second subset of tasks.

13. A packet processing system as recited in claim 12, wherein the third subset of tasks are qualitatively different than the first subset of tasks.

14. A packet processing system as recited in claim 12, wherein the third subset of tasks are quantitatively different than the first subset of tasks.

15. A packet processing system for receiving and processing a data packet from a media access control device or framer, wherein the packet processing system is coupled with a switching device for transmitting the packet to another network device after it is processed by the packet processing system, the packet processing system comprising:

a programmable traffic classifier arranged to perform a plurality of traffic classification tasks on the packet that are selected from a group of tasks consisting of classifying the packet, filtering the packet, encrypting or decrypting the packet, and editing the packet, and scheduling the packet, the first processor being further arranged to attach a data manipulation header to the packet; and

a traffic manager arranged to receive the packet and perform a plurality of traffic management tasks on the packet that are selected from a group of tasks consisting of providing a plurality of queues for routing the packet, quality of service tasks, network scheduling, congestion handling, and queue reordering functions, the management tasks being based on the data manipulation header.

16. A packet processing system as recited in claim 15, further comprising a monitor processor arranged to perform a plurality of monitoring tasks that include tasks selected from a group of tasks consisting of flow classification, flow based statistical database management, and complex filtering and data compaction.

17. A packet processing system as recited in claim 16, wherein the classification, management, and monitoring tasks are individually programmable.

18. A packet processing system as recited in claims 16 or 17, wherein the monitor processor is further arranged to perform its monitoring tasks in parallel to classification and/or the management tasks.

19. A packet processing system comprising:

a traffic classifier arranged to receive an incoming packet and associate classifier information with the received packet, the classifier information being indicative of or useful in at least one of desired routing and prioritization of the received packet, the traffic classifier including,

a first stream processor arranged to perform a first set of tasks on the received data packet and to associate a first header with the packet, the first header being based on information obtained while performing the first subset of tasks, a queue structure arranged to receive the packet,

a search engine for looking up data associated with information in the packet based on the first header and create a second header that is associated with the packet in the queue structure; and

a traffic manager having a multiplicity of queues associated therewith for queuing packets to be transmitted to another device, the traffic manager being arranged to receive packets from the traffic classifier and insert them in an appropriate queue based at least in part upon the classifier information associated with the packet by the traffic classifier.

20. A packet processing system as recited in claim 19 further comprising a second stream processor arranged to perform a second set of tasks associated with the packet based at least in part on the second header.

21. A packet processing system as recited in claim 20 wherein the second stream processor is further arranged to modify the second header based at least in part on information obtained while performing the second subset of tasks.

22. A packet processing system as recited in claim 21, wherein the information obtained while performing the second set of tasks include extracting information from the received data packet based at least in part on the second header.

23. A packet processing system as recited in any of claims 19-22 wherein the queue structure is arranged to receive the packet as a processed packet from the first stream processor.

24. A packet processing system as recited in any of claims 19-22 wherein the queue structure is arranged to receive the packet in parallel with the first stream processor.

25. A packet processing system as recited in any of claims 19-24 further comprising:

a transmit queue structure having a plurality of linked queues for holding a transmission packet for transmission to a media access control device; and

a second stream processor arranged to perform a second set of tasks on the transmission packet prior to transmitting it to the media access control device.

26. A packet processing system as recited in claim 25, wherein the second set of tasks of the second stream processor include prioritizing the transmission packet into a selected linked queue.

27. A packet processing system as recited in any of claims 19-26, further comprising a media access control bus interface for interfacing with a media access control device.

28. A packet processing system as recited in any of claims 19-27, wherein the tasks of the first stream processor include extracting information from the received data packet and the first header is based on in part on the extracted information.

29. A packet processing system as recited in claim 28, wherein the first header includes a pointer to a portion of the received data packet.

30. A packet processing system as recited in claim 28, wherein the first header includes a hashed portion of the received data packet.

31. A packet processing system as recited in any of claims 19-30, wherein the second header is a data manipulation layer header that indicates how to handle the received data packet.

32. A packet processing system as recited in any of claims 19-31, wherein the first header is a key header that includes the source and destination address of the packet.

33. A packet processing system as recited in cl any of claims 19-32, wherein the search engine is arranged to look up data within a database, the looked up data being associated with the first header, and the second header is based on at least one of information from the first header and the looked up data.

34. A packet processing system as recited in claim 33, wherein the looked up data within the database includes information selected from a group consisting of policy information, queue priority information, and security information that is associated with a destination of the received data packet.

35. A packet processing system as recited in claim 34, wherein the first header includes a pointer to the destination of the received data packet.

36. A packet processing system comprising:

a traffic classifier arranged to receive an incoming packet and associate information indicative of or useful in at least one of routing and prioritizing the packets, the traffic classifier including:

a stream processor arranged to extract information from the packet,

a search engine arranged to look up data based on information extracted by the stream processor and to associate such looked up data with the packet; and

a traffic manager having a multiplicity of queues associated therewith for queuing packets to be transmitted to another device, the traffic manager being arranged to receive packets from the traffic classifier and insert them in an appropriate queue based at least in part upon the information associated with the packet by the traffic classifier.

37. A data processing system as recited in claim 36, wherein the traffic classifier further includes a post editor arranged to modify the packet and/or information associated with the packet.

38. A router comprising:

a switching device;

a plurality nodes that are coupled to the switching device, each node including a packet processing system as recited in claim 19, a media access controller and a physical layer controller.

39. A router as recited in claim 38 further comprising a management processor system and a management bus, each packet processor being coupled to the management bus.

40. A router as recited in claims 38 or 39, further comprising a memory device associated with each node.

41. A router as recited in claims 39 or 40, wherein the management processor system is arranged to program at least one of the first stream processor, search engine, and traffic manager.

42. A router as recited in claim 41, wherein the programming is selected to

accomplish a result selected from a group consisting of balancing load, balancing data flow, minimizing traffic congestion, optimal and programmable sharing of bandwidth, profiling of traffic, and quality of service.

43. A router as recited in any of claims 38-42, wherein each node has an associated processor and each processor is capable of sharing information with another processor through the switching device.

44. A method for processing packet data received from a media access layer, the processing being performed in-line while streaming packets to an upper layer, comprising:

loading an instruction set for custom programming the processing of packet data received from the media access layer;

receiving the packet data within a first processor

extracting information from the packet data based at least in part on the instruction set;

creating a first data structure based at least in part on the extracted information;

associating the first data structure with the packet data before the packet is streamed to a second processor;

extracting information from one or more databases based on the first data structure;

creating a second data structure based at least in part on the first data structure and information extracted from the one or more databases;

associating the second data structure with the packet data before the packet is streamed to a third processor;

receiving the packet data within the third processor having a plurality of queues for routing data to various devices; and

linking the data packet to a particular queue of the third processor based at least in part on the second data packet.

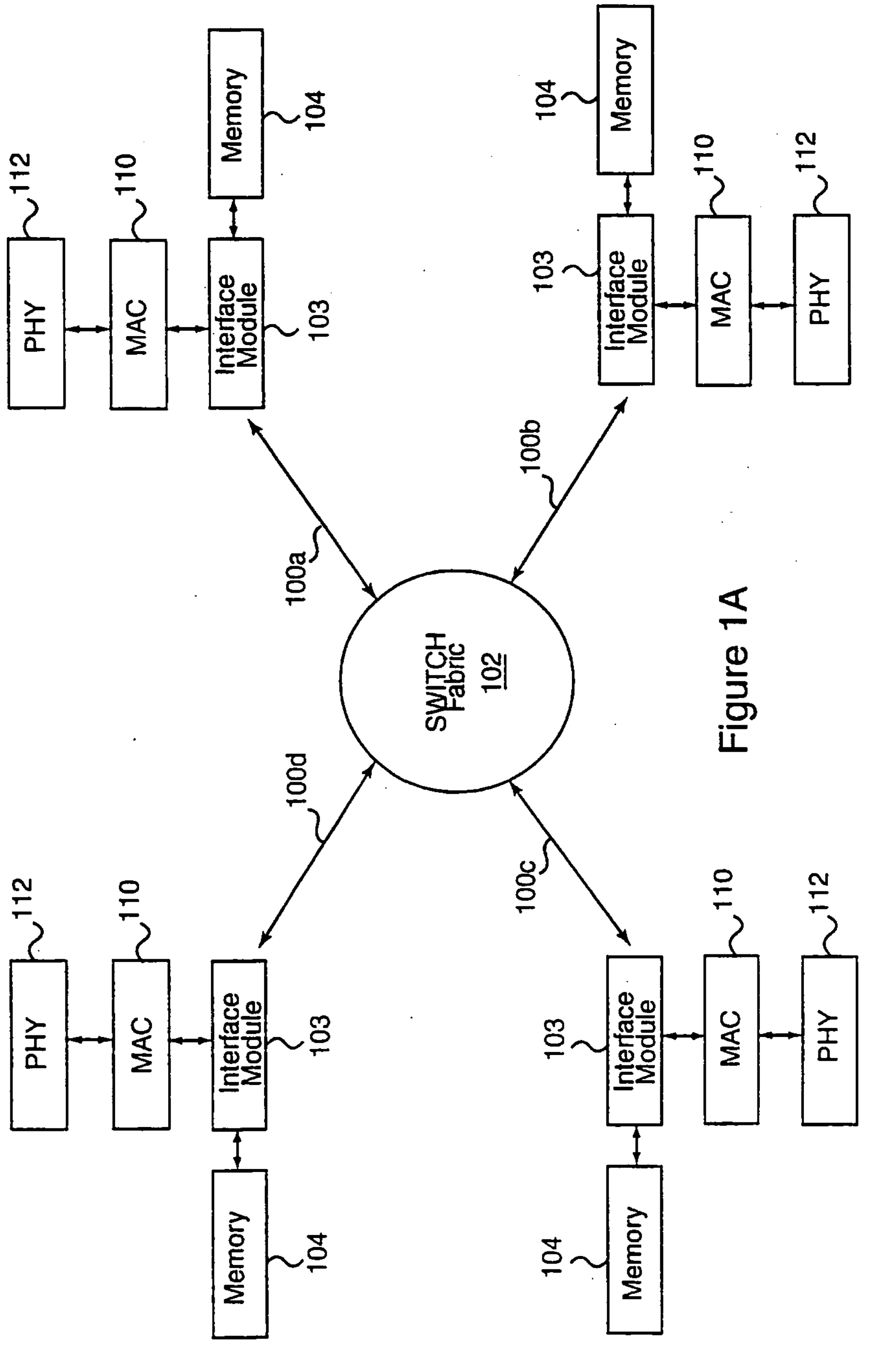

SWITCH
Fabric
102
100a
100b
100c
100d
Interface Module
103
MAC
110
PHY
112
Memory
104

Figure 1A

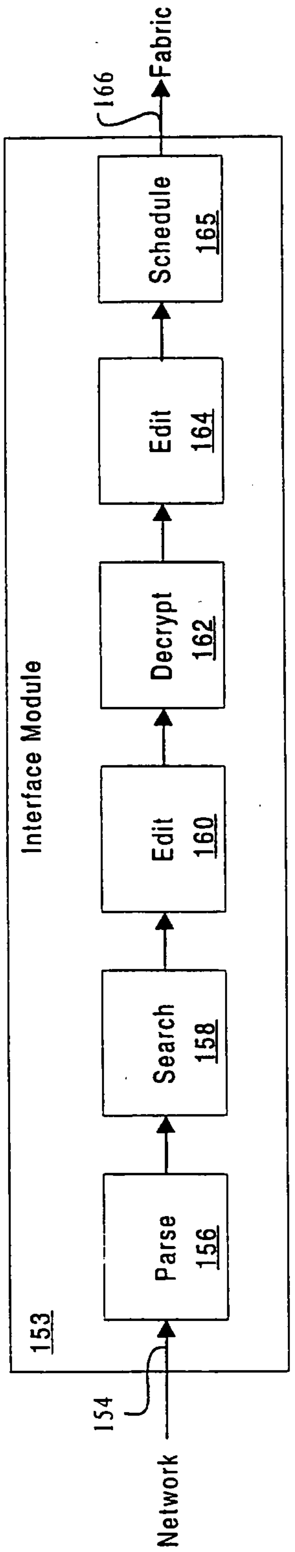


Figure 1B

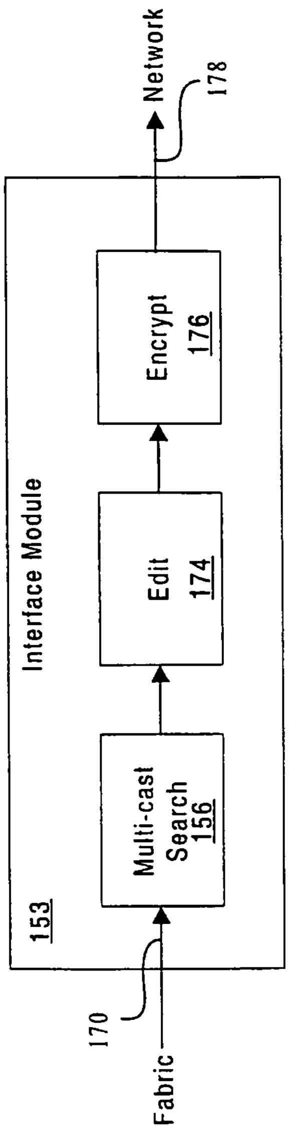


Figure 1C

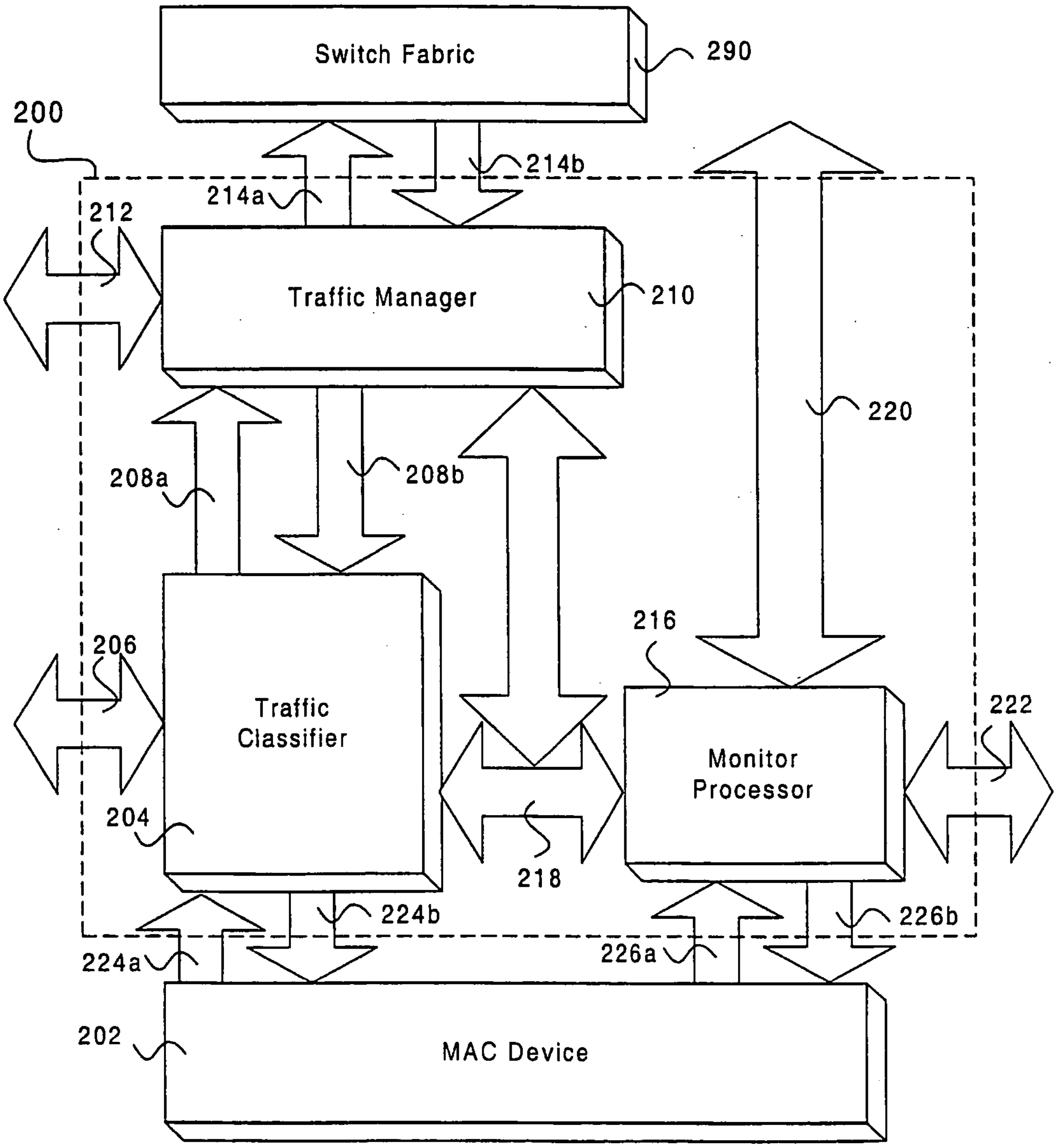

Switch Fabric
290
200
214b
214a
212
Traffic Manager
210
220
208a
208b
216
206
Traffic Classifier
222
Monitor Processor
204
218
224b
226b
224a
226a
202
MAC Device

Figure 2A

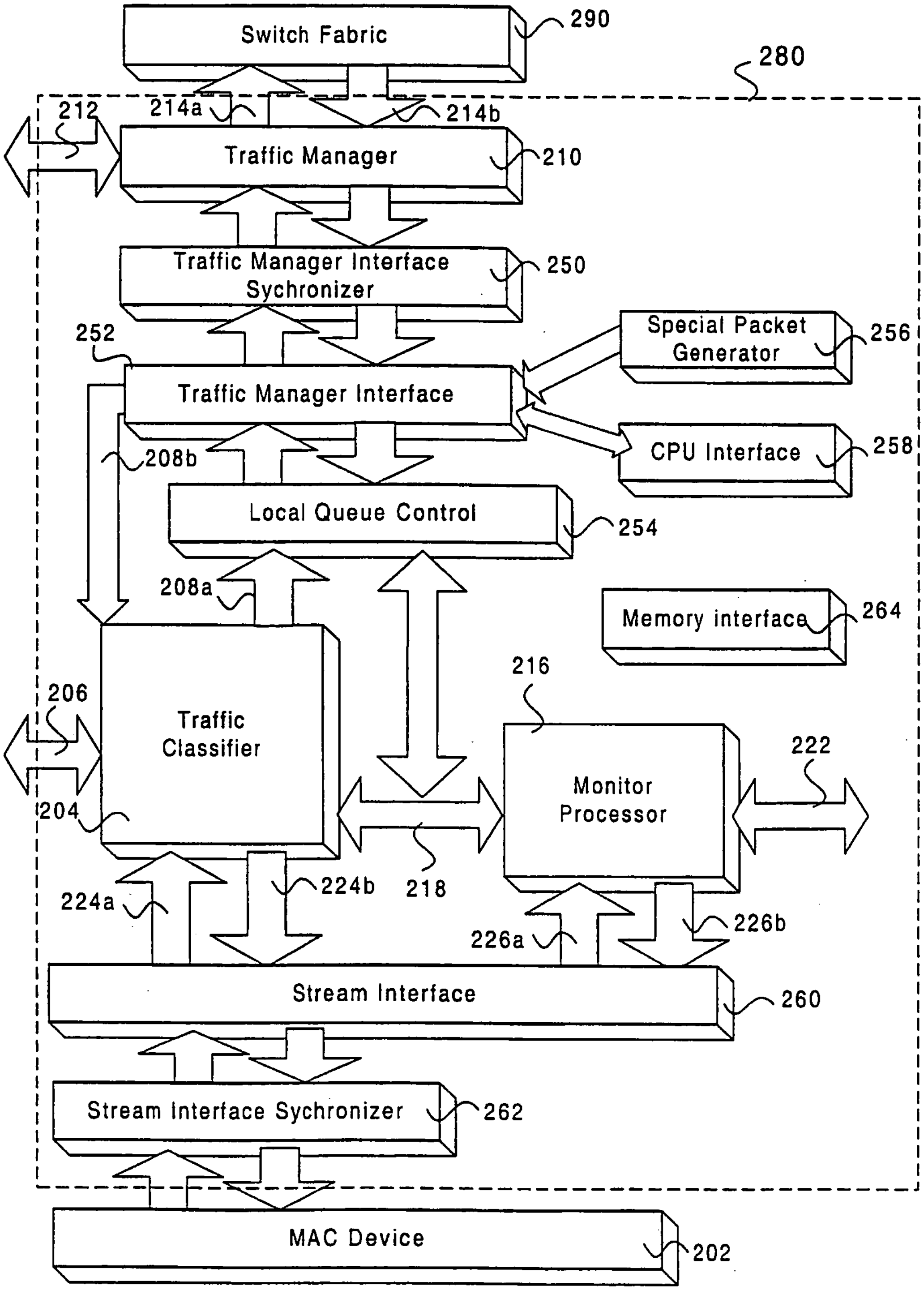

Switch Fabric
290
280
212
214a
214b
Traffic Manager
210
Traffic Manager Interface Sychronizer
250
Special Packet Generator
256
252
Traffic Manager Interface
CPU Interface
258
208b
Local Queue Control
254
208a
Memory interface
264
216
206
Traffic Classifier
Monitor Processor
222
204
224a
224b
218
226a
226b
Stream Interface
260
Stream Interface Sychronizer
262
MAC Device
202

Figure 2B

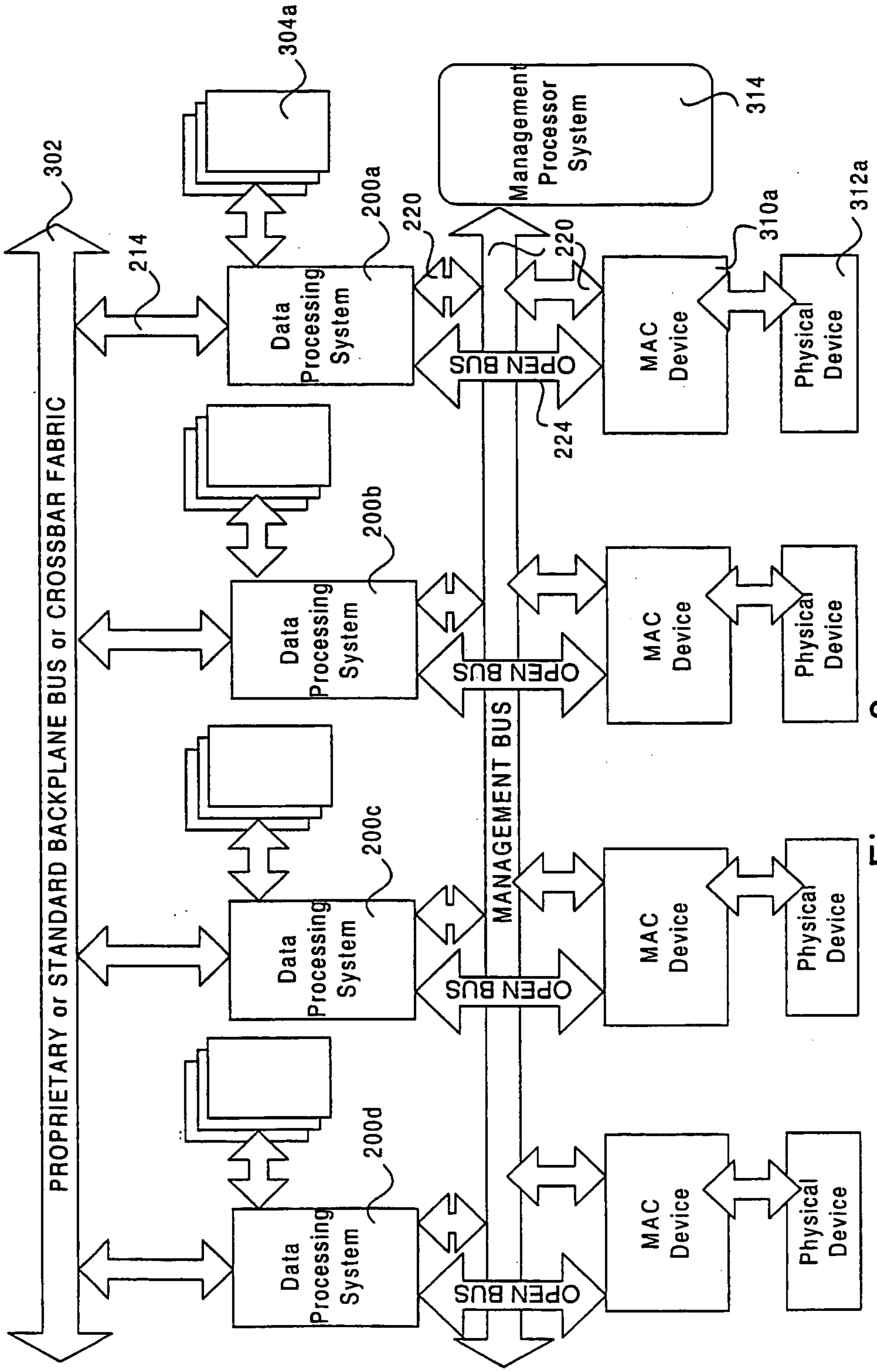

PROPRIETARY or STANDARD BACKPLANE BUS or CROSSBAR FABRIC
302
214
304a
Data Processing System
200a
200b
200c
200d
220
220
220
Management Processor System
314
MANAGEMENT BUS
OPEN BUS
224
MAC Device
310a
Physical Device
312a

Figure 3

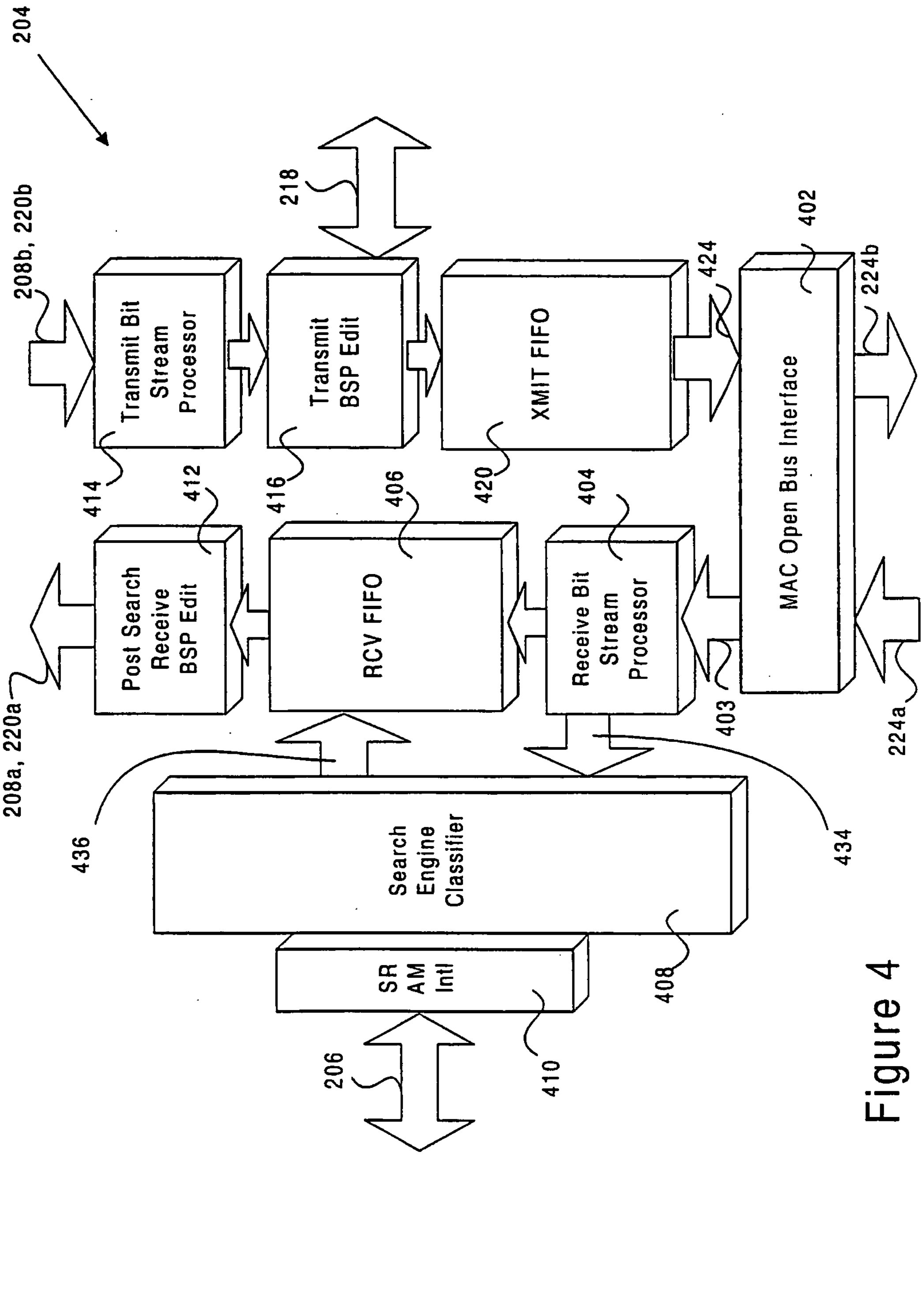
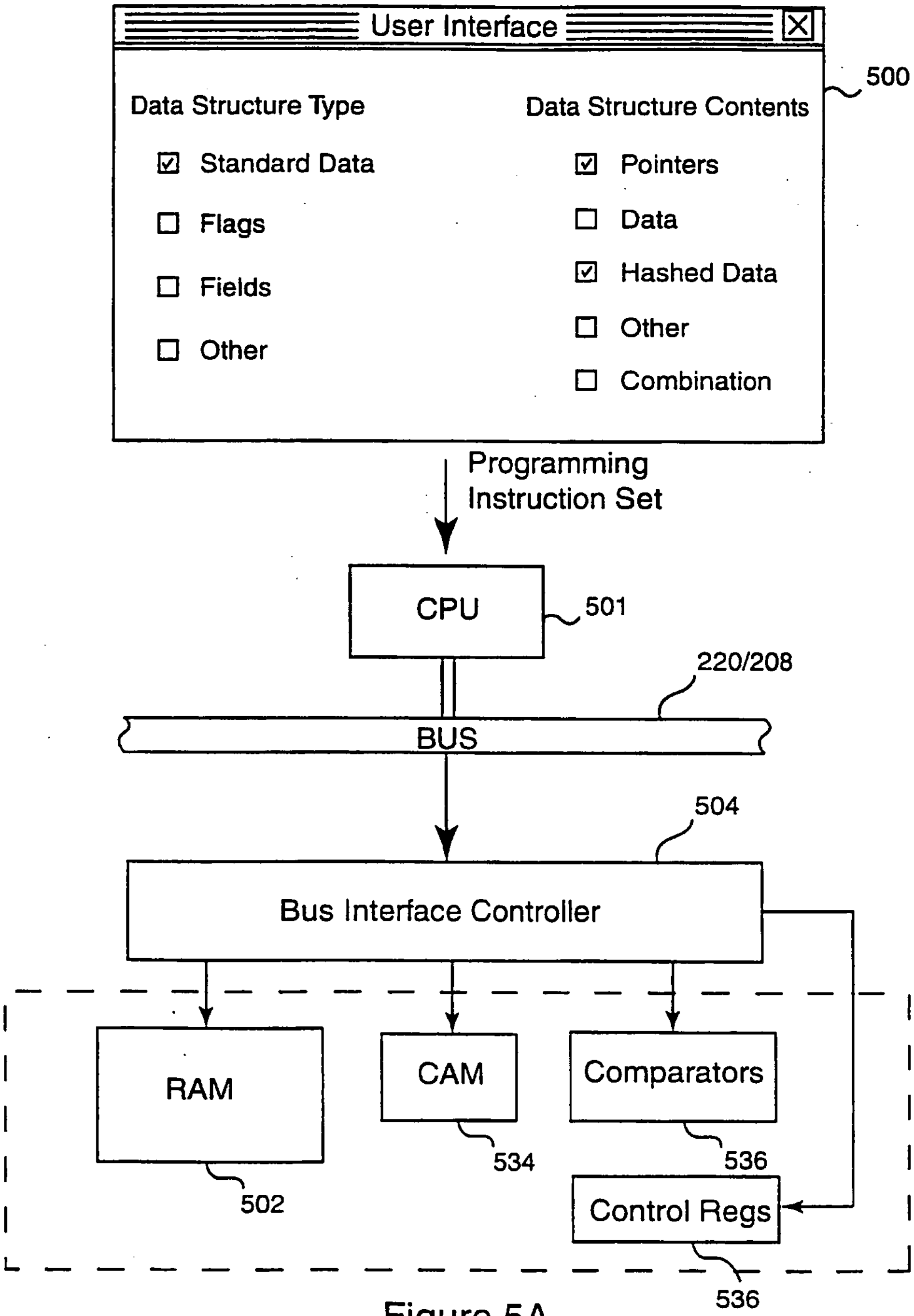

204
208b, 220b
Transmit Bit Stream Processor
414
Transmit BSP Edit
416
218
XMIT FIFO
420
424
MAC Open Bus Interface
402
224b
224a
403
Receive Bit Stream Processor
404
RCV FIFO
406
Post Search Receive BSP Edit
412
208a, 220a
436
434
Search Engine Classifier
408
SR AM Intf
410
206

Figure 4



User Interface ☒

Data Structure Type

☑ Standard Data

□ Flags

□ Fields

□ Other

Data Structure Contents

☑ Pointers

□ Data

☑ Hashed Data

□ Other

□ Combination

Figure 5A

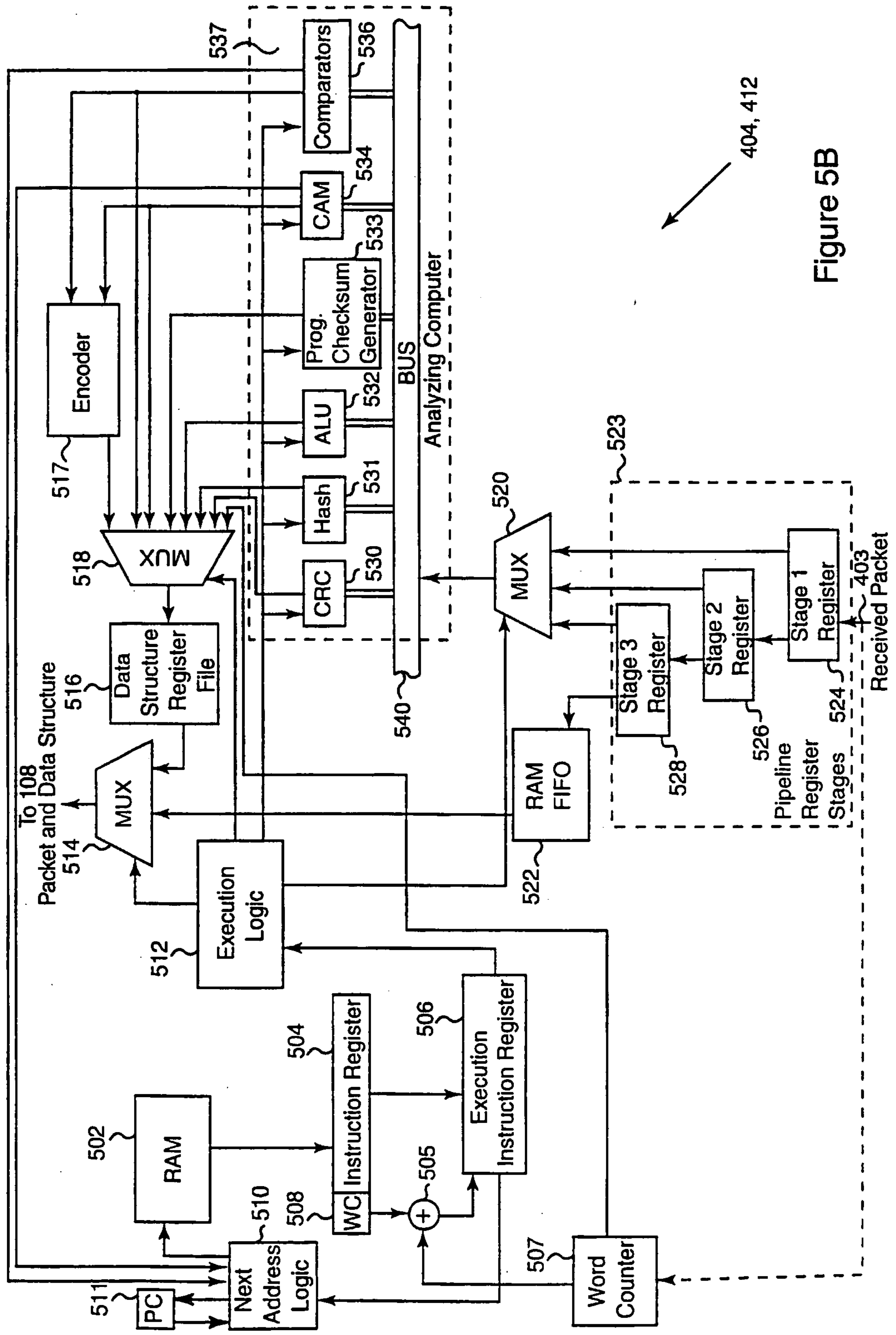

537
Comparators
536
CAM
534
Prog. Checksum Generator
533
ALU
532
Hash
531
CRC
530
BUS
Analyzing Computer
540
404, 412
Encoder
517
MUX
518
Data Structure Register File
516
To 108 Packet and Data Structure
MUX
514
Execution Logic
512
MUX
520
523
Stage 1 Register
524
Stage 2 Register
526
Stage 3 Register
528
403
Received Packet
Pipeline Register Stages
RAM FIFO
522
RAM
502
Instruction Register
504
WC
508
505
+
Execution Instruction Register
506
Next Address Logic
510
PC
511
Word Counter
507

Figure 5B

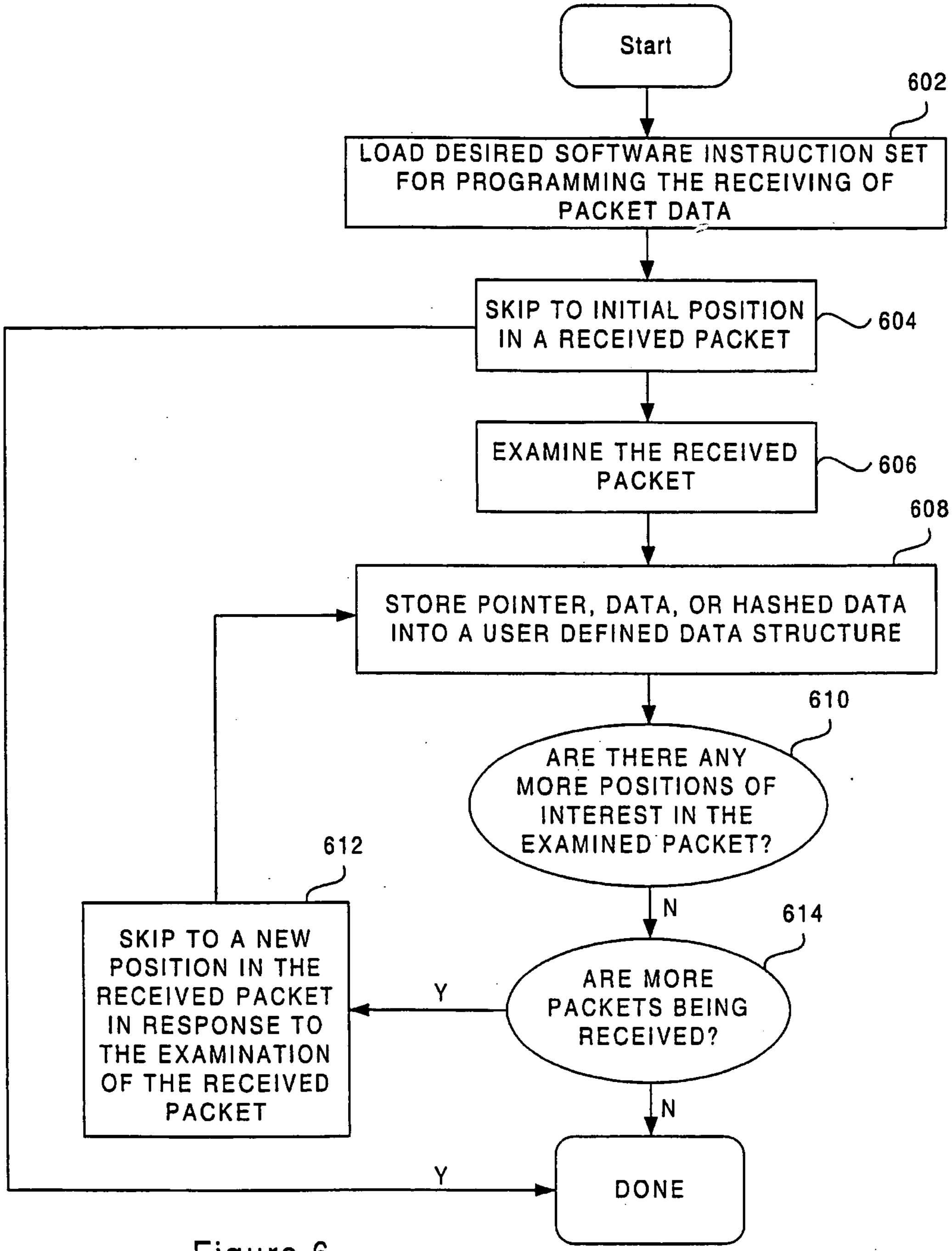

Start
602
LOAD DESIRED SOFTWARE INSTRUCTION SET FOR PROGRAMMING THE RECEIVING OF PACKET DATA
SKIP TO INITIAL POSITION IN A RECEIVED PACKET
604
EXAMINE THE RECEIVED PACKET
606
608
STORE POINTER, DATA, OR HASHED DATA INTO A USER DEFINED DATA STRUCTURE
610
ARE THERE ANY MORE POSITIONS OF INTEREST IN THE EXAMINED PACKET?
N
612
SKIP TO A NEW POSITION IN THE RECEIVED PACKET IN RESPONSE TO THE EXAMINATION OF THE RECEIVED PACKET
Y
614
ARE MORE PACKETS BEING RECEIVED?
N
Y
DONE

Figure 6

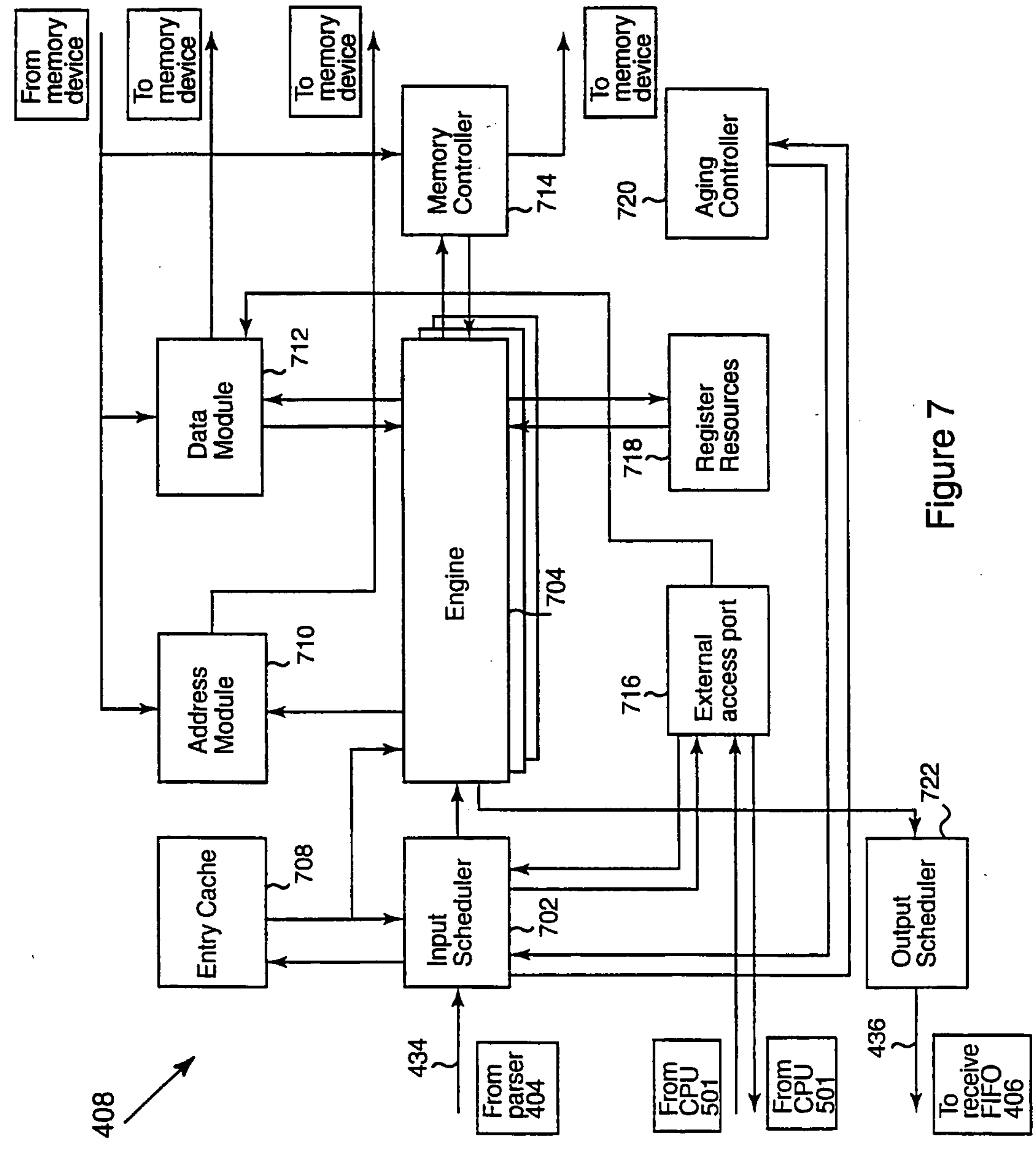

From memory device
To memory device
To memory device
To memory device
Memory Controller
714
720
Aging Controller
712
Data Module
Engine
704
718
Register Resources
710
Address Module
716
External access port
708
Entry Cache
702
Input Scheduler
722
Output Scheduler
434
From parser 404
From CPU 501
From CPU 501
436
To receive FIFO 406
408

Figure 7

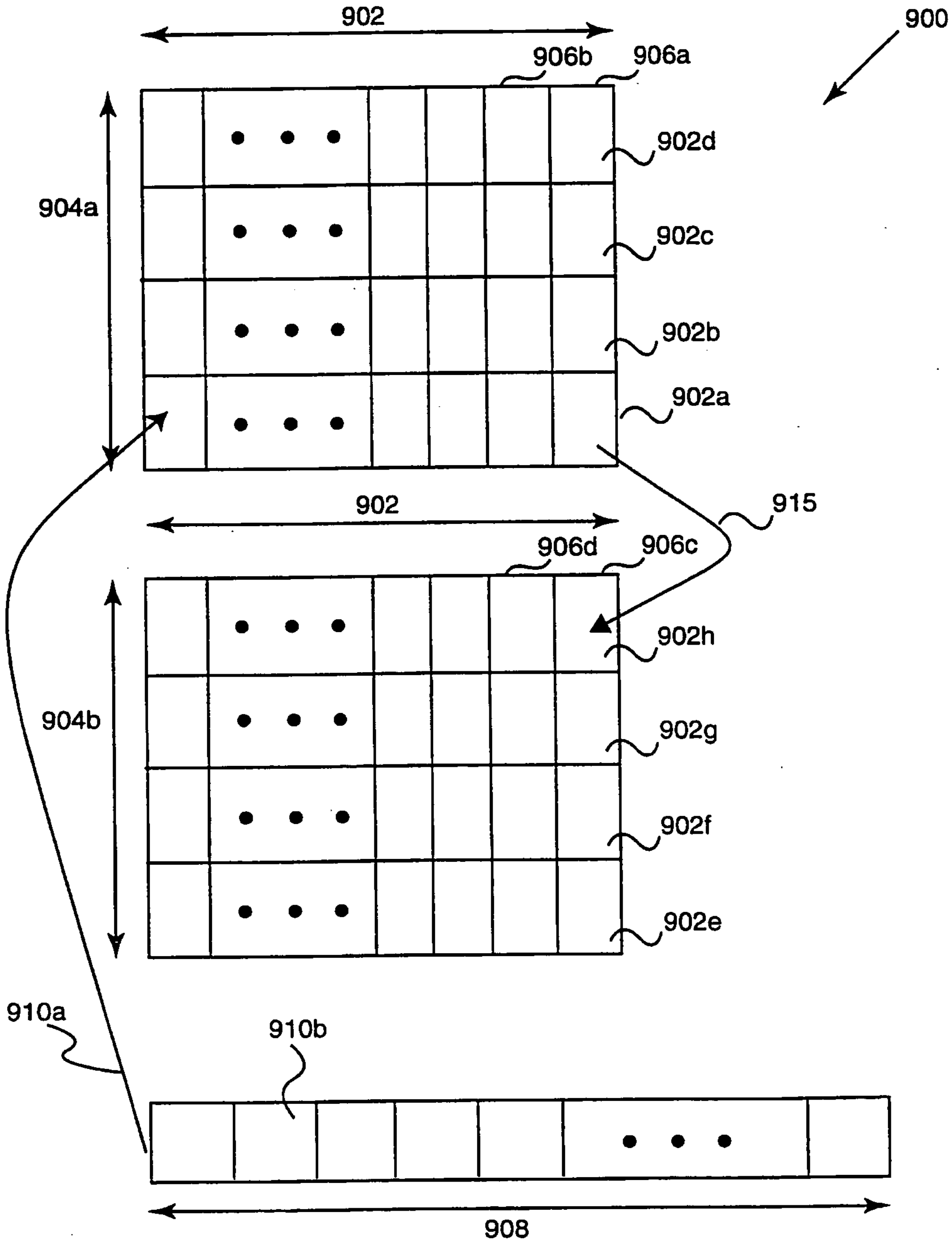

902
900
906b
906a
902d
904a
902c
902b
902a
902
915
906d
906c
902h
904b
902g
902f
902e
910a
910b
908

Figure 8

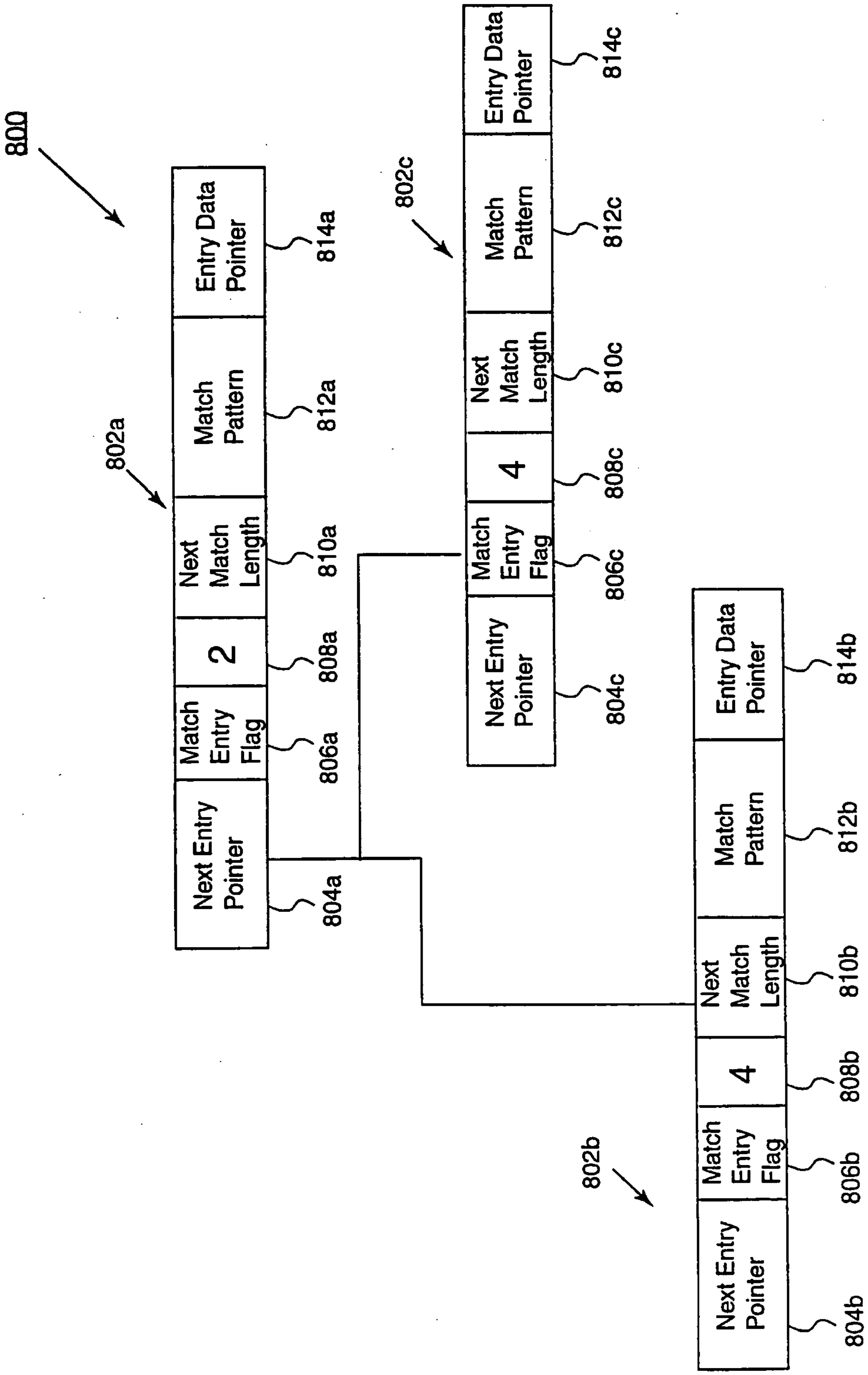

800
802a
Next Entry Pointer
Match Entry Flag
2
Next Match Length
Match Pattern
Entry Data Pointer
804a
806a
808a
810a
812a
814a
802c
Next Entry Pointer
Match Entry Flag
4
Next Match Length
Match Pattern
Entry Data Pointer
804c
806c
808c
810c
812c
814c
802b
Next Entry Pointer
Match Entry Flag
4
Next Match Length
Match Pattern
Entry Data Pointer
804b
806b
808b
810b
812b
814b

Figure 9